

ORACLE®

# Real-World Performance Training

Core Database Performance

Real-World Performance Team



ORACLE®

ORACLE®  
REAL-WORLD PERFORMANCE

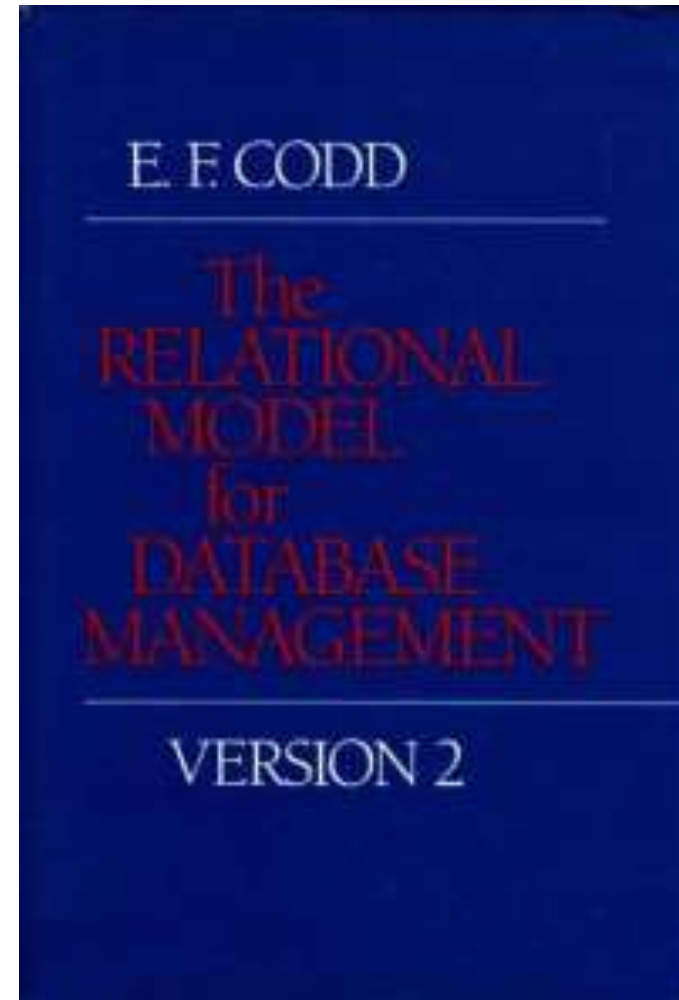
# Agenda

- 1 Computer Science Basics
- 2 Schema Types and Database Design
- 3 Database Interface
- 4 DB Deployment and Access Options
- 5 Application Algorithms
- 6 Resource Management

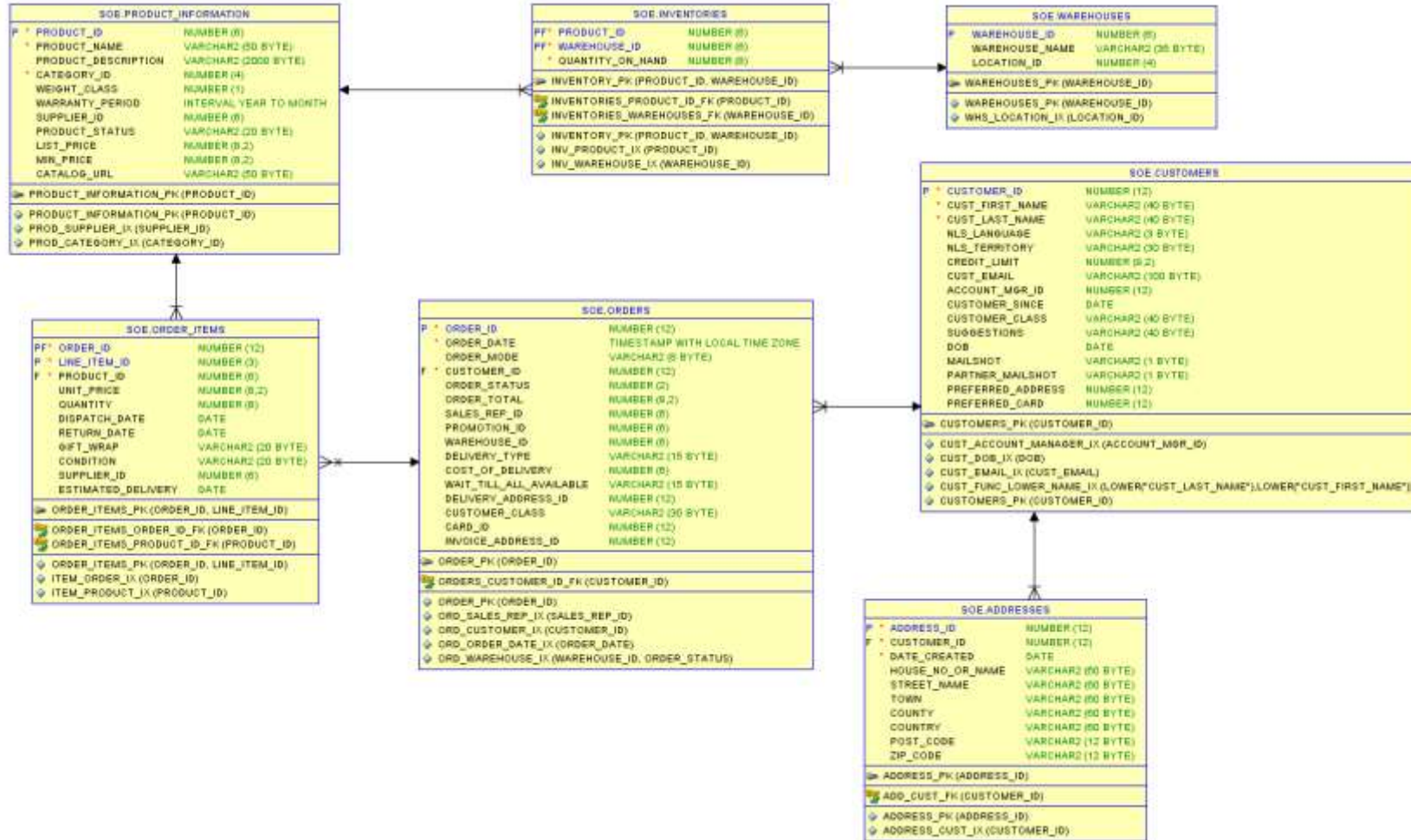
# Agenda

- 1 Computer Science Basics
- 2 Schema Types and Database Design
- 3 Database Interface
- 4 DB Deployment and Access Options
- 5 Application Algorithms
- 6 Resource Management

# Schema Types And Database Design



# Third Normal Form



# Third Normal Form

## Key Design Characteristics

- No Duplication of data—data only stored once
- All columns in an entity are directly dependent on the primary key
- Reflects “Real-World Data” model
- Requires joins to retrieve the data
- Rigour and Discipline to produce
- Care must be taken not to go into Analysis/Paralysis

# Third Normal Form

## Definition

*Each non-key attribute "must provide a fact about the key, the whole key, and nothing but the key."*

*Requiring that non-key attributes be dependent on "the whole key" ensures that a table is in 2NF; further requiring that non-key attributes be dependent on "nothing but the key" ensures that the table is in 3NF."*

[http://en.wikipedia.org/wiki/First normal form](http://en.wikipedia.org/wiki/First_normal_form)

[http://en.wikipedia.org/wiki/Second normal form](http://en.wikipedia.org/wiki/Second_normal_form)

[http://en.wikipedia.org/wiki/Third normal form](http://en.wikipedia.org/wiki/Third_normal_form)

# Denormalization

- Sometimes at Database Design time it is necessary to denormalize some data
- Denormalization involves movement of columns, often duplicating or summarising data in other entities
- Denormalization is usually done for performance reasons, for example to avoid joins/aggregations for reporting purposes

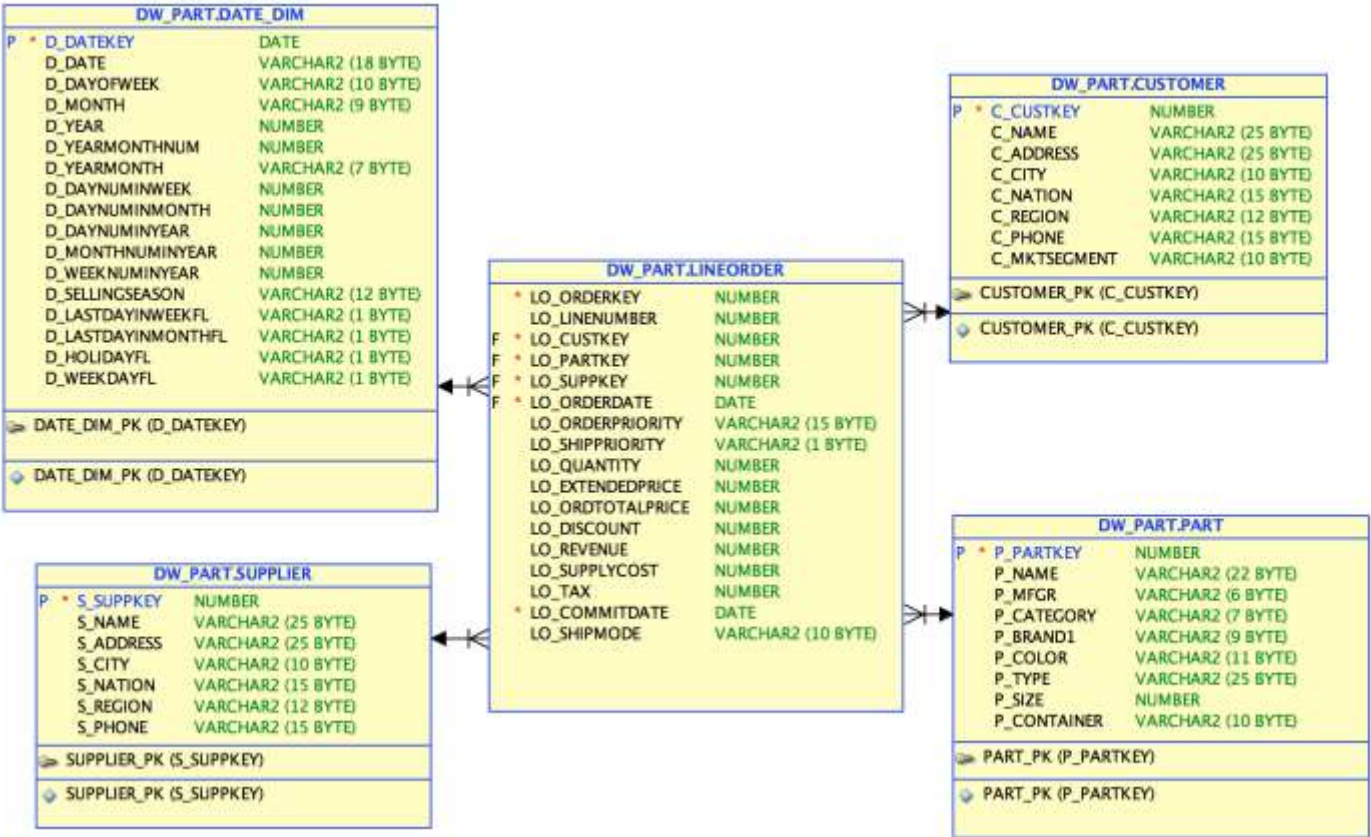
# Denormalization

## Issues

- If the data is duplicated it has to be maintained in more than one place
- Often data is denormalized to get around perceived expensive SQL (e.g. summations and joins)
- Tends to be done a lot more often than necessary due to poor root cause analysis

# Dimensional Model

## Star and Snowflake schemas

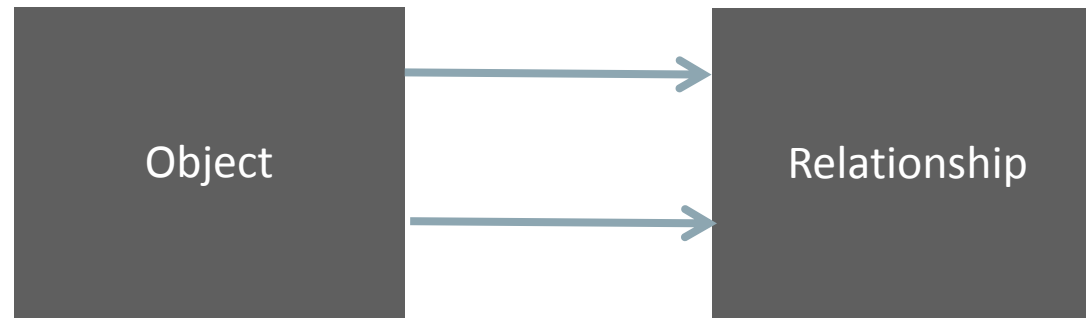


# Dimensional Model

## Characteristics

- Usually used in analytical applications
- Single “FACT” table which contains details of the main transactions denormalized into a single table (e.g. Order + Order\_Items combined into SALES)
- Dimension tables contain the data associated with the dimension not in the facts
- Constraints define relationships
- Data usually appended and not updated
- FACT table usually dwarfs the other tables in terms of number of rows and columns

# Abstract “Meta-Model”



# Abstract Meta-Model

## Characteristics

- Easy to model
- Difficult to implement and debug
- Any query usually joins all the tables at least once and often back to itself many times
- Difficult, if not impossible, to scale

# “Everything is a column” Design

Table structure like:

| Column     | Desc          |
|------------|---------------|
| Id         | Number        |
| Value_type | Varchar2(32)  |
| Value      | Varchar2(255) |

Similar to the “flexfields” used in some Oracle Applications, ultimately flexible but difficult to get any performance out of it

# Other Models

- Key Value Pair modelling (NoSQL in SQL) – single key field followed by a payload which can be JSON/XML/AVRO etc. etc.
- Usually symptom of developers doing database design and seeing things as persistence put and get operators
- Good for one thing – put, get and update and not harnessing the power of the Database Language, Hardware or investment in data.

# Hybrid Designs

- Most designs use elements from all previous models
- Potentially optimised for performance and flexibility but poor choices of Hybrid design may combine the worst of all techniques and not the best.

# Agenda

- 1 Computer Science Basics
- 2 Schema Types and Database Design
- 3 Database Interface**
- 4 DB Deployment and Access Options
- 5 Application Algorithms
- 6 Resource Management

# Database Interface



# High Performance Applications

## The Challenge

- The impact of how application code impacts system performance should never be underestimated
  - This fact has been known for a long time
  - It has been ignored for almost as long
- Education of developers on the correct way to write code is a continuous, repeating activity
  - New developers graduate every year!
- Poor coding techniques combined with classic programmer bugs can render investments in the system worthless

# Bad Performance

## Observations

- A problematic application has never met performance objectives
- Response time and throughput are poor
- Everybody blames the database
  - DBA sees no real issues
  - DBA suggests adding more connections to drive up workload
- The system must be able to execute a minimum of 35,000 transactions per second to survive Thanksgiving and Black Friday

# Parsing Demo

Cursors and Connections

ORACLE<sup>®</sup>  
RWP Video



# Parsing Demo

## Bad Performance with Logons



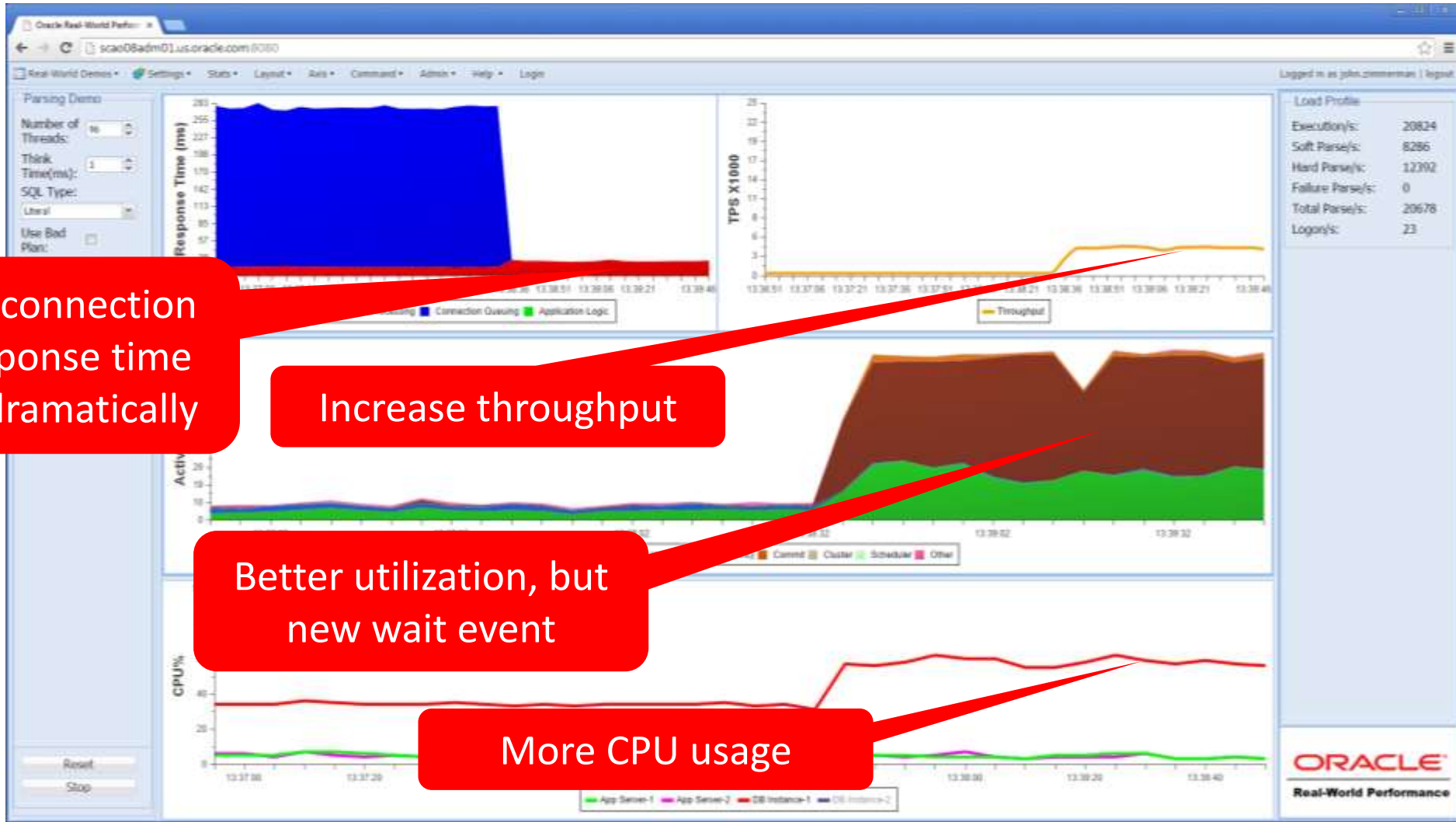
Poor response time

Low transaction rate

Cannot make the system busy

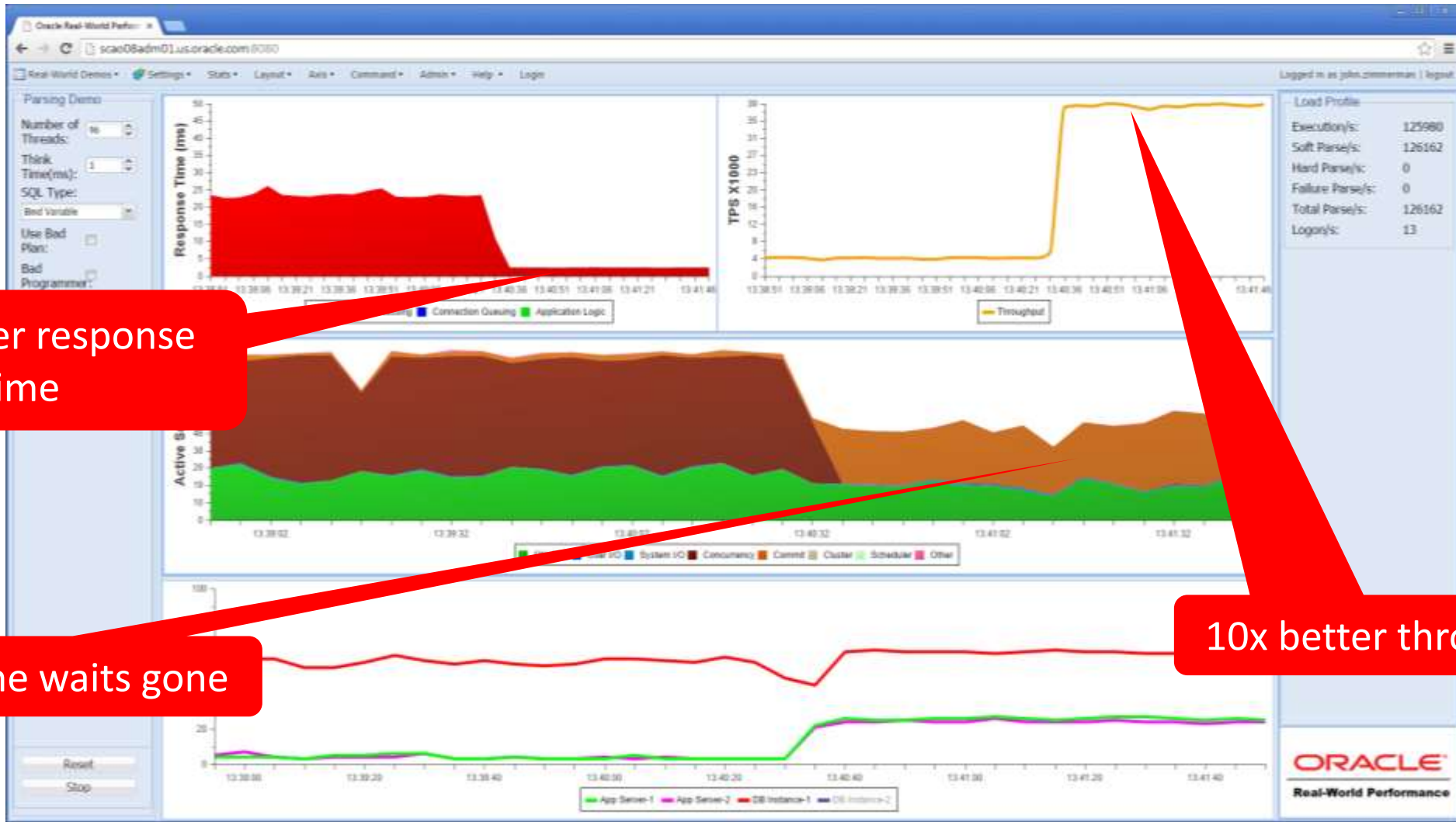
# Parsing Demo

## Connection Pools and Hard Parse



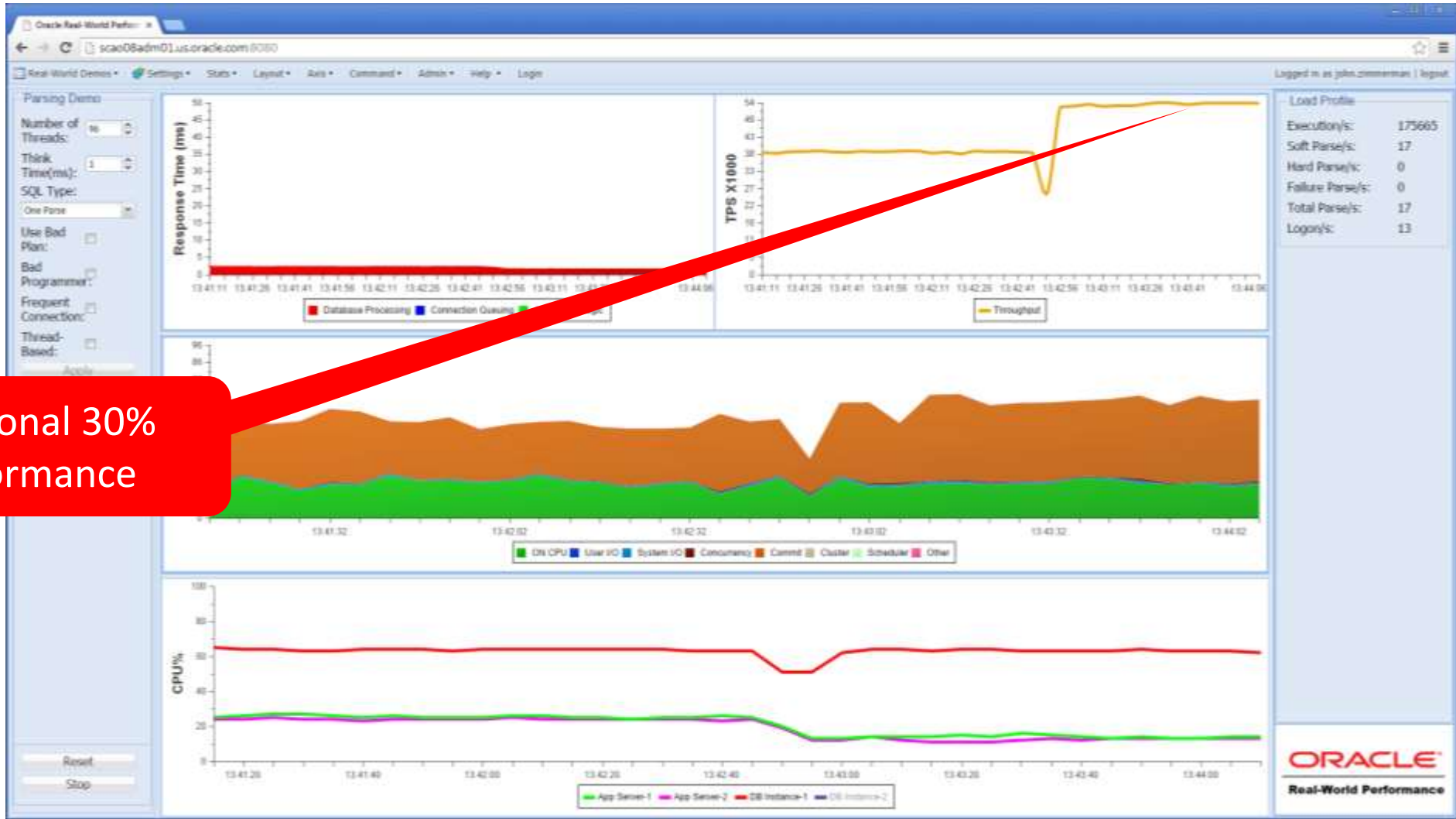
# Parsing Demo

## Bind Variables and Soft Parse



# Parsing Demo

## Shared Cursors and One Parse

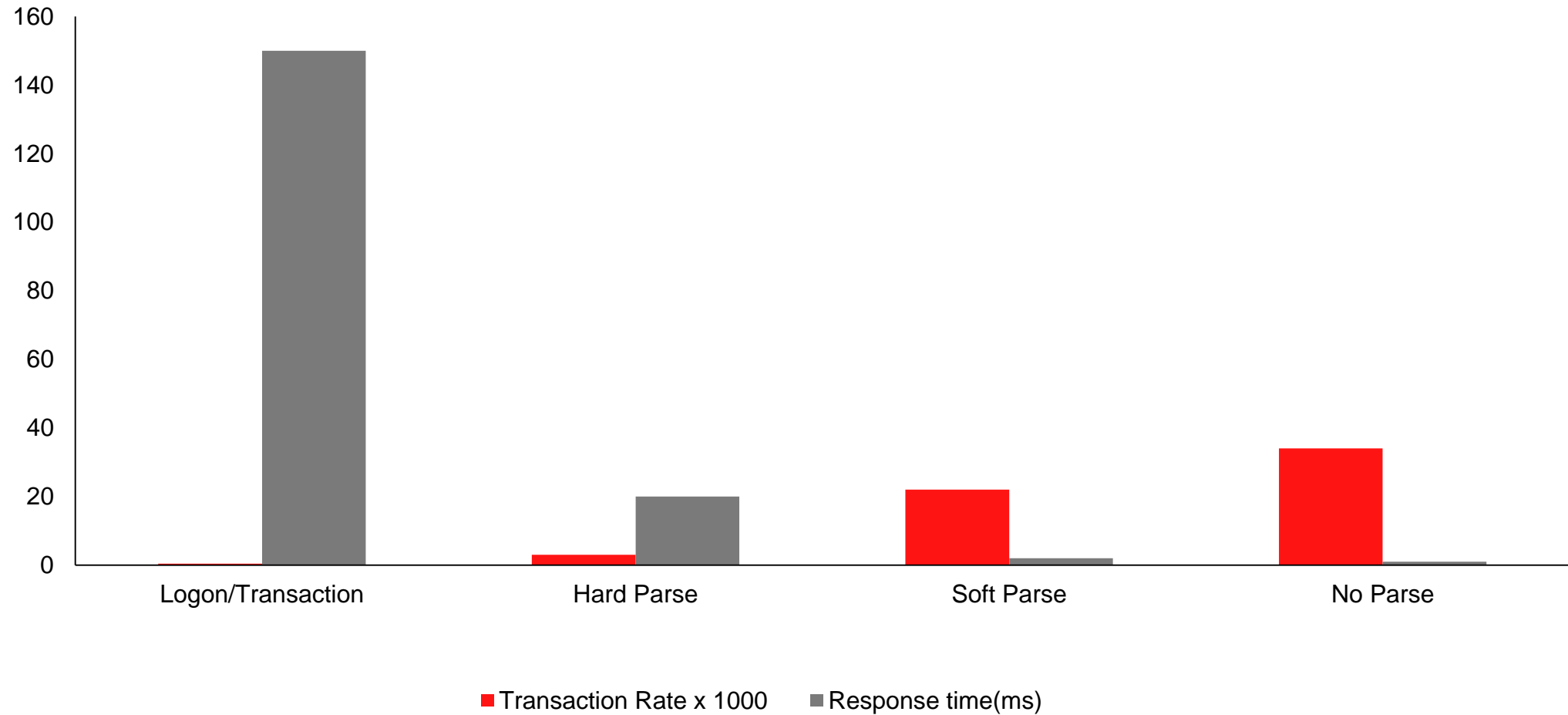


Additional 30% performance



# Parsing Demo

## Incorrect Use of Sessions and Cursors

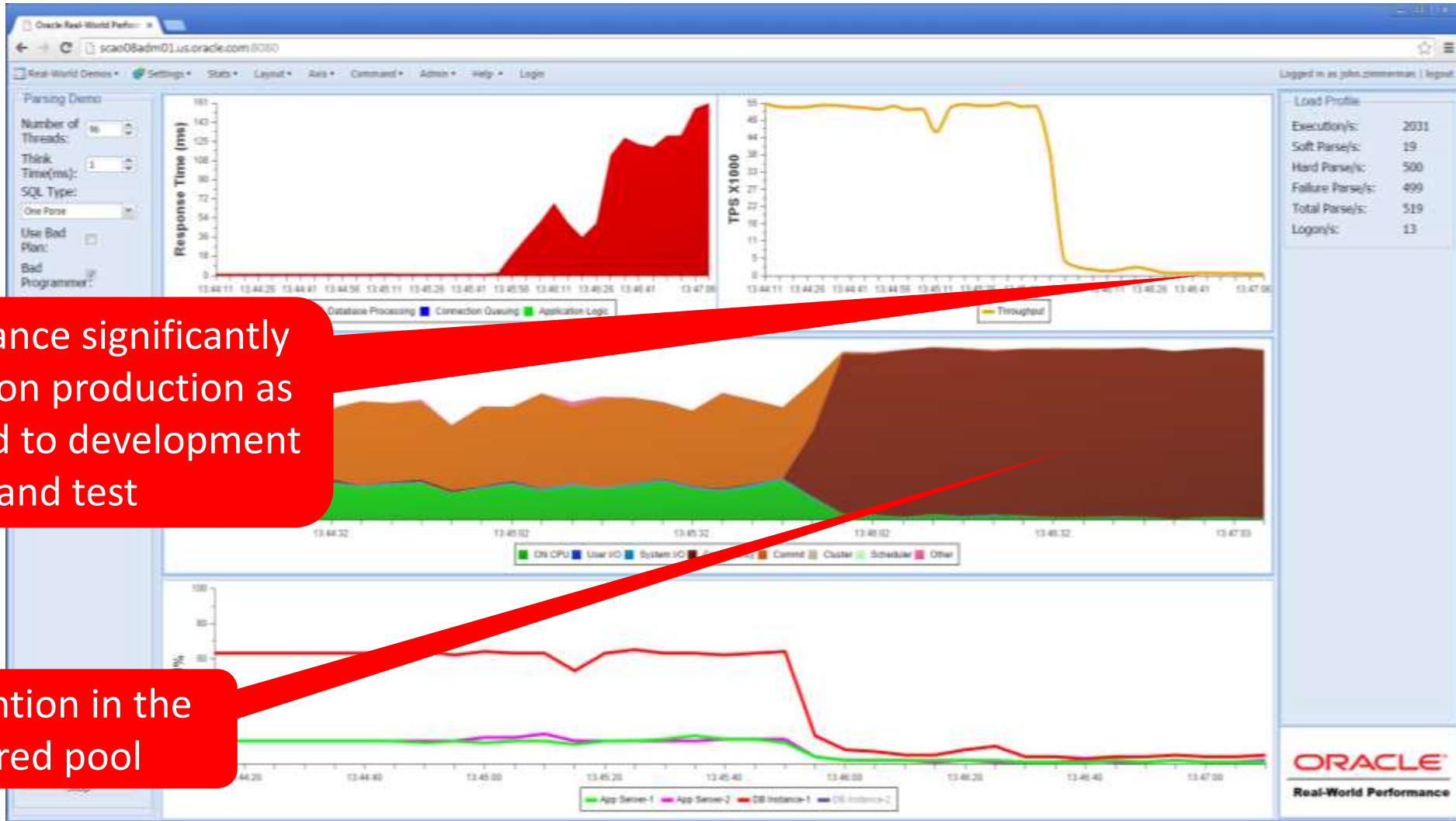


# Parsing Demo

## Observations

- Both the development and DBA teams are confused.
  - Performance in the development and test systems is as anticipated
  - Performance in production is nowhere near level of test system
  - DBAs see shared pool contention but developers have coded diligently to ensure no parsing
  - Development has confirmed the same code is running in both test and production

# Invalid SQL Performance Data



Performance significantly reduced on production as compared to development and test

Contention in the shared pool

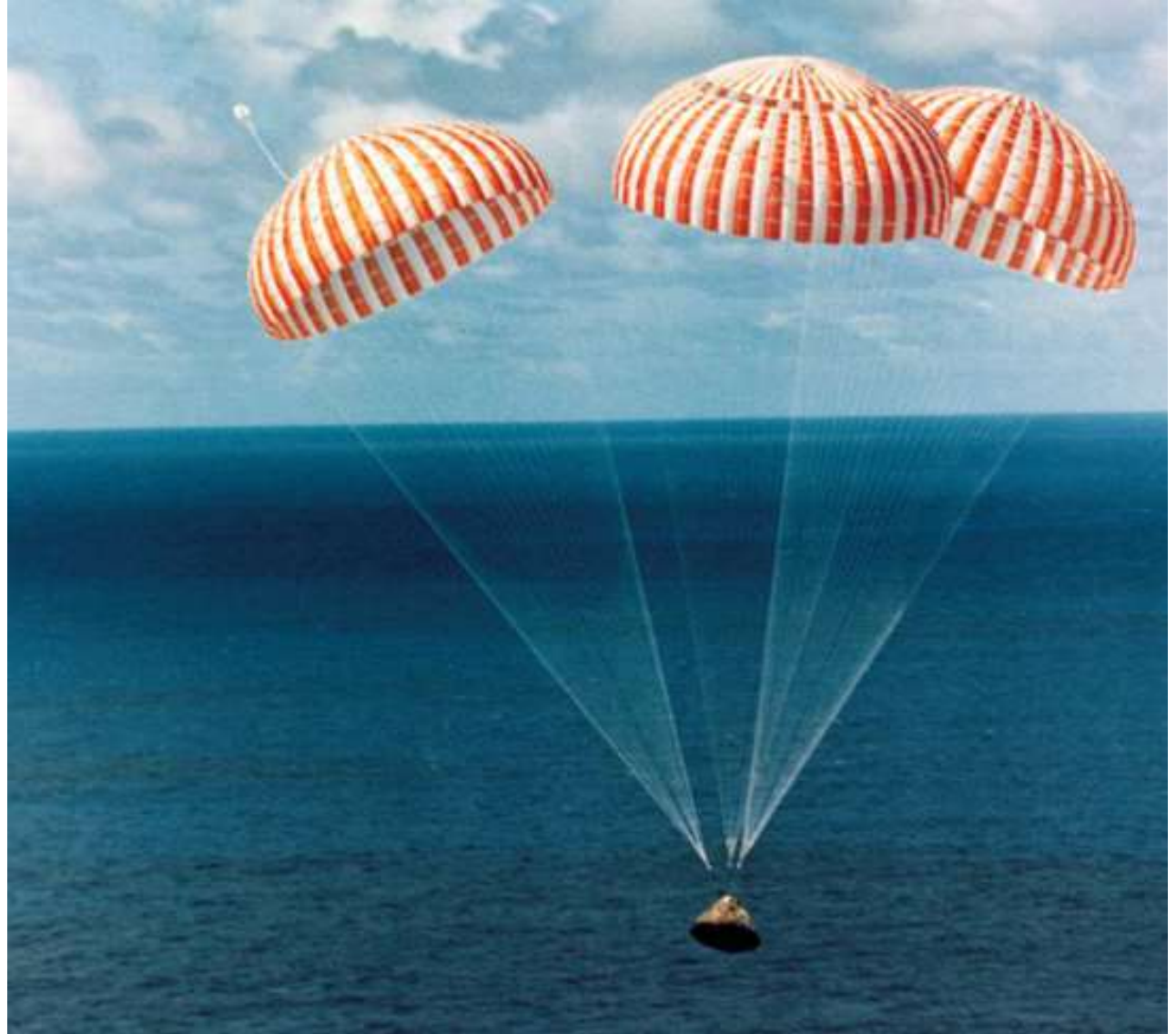
# Invalid SQL

- A page refresh trigger attempts to set a session-level initialization parameter to enable a diagnostic patch that is not installed in production
  - This results in a failed parse
- All users are frequently attempting to parse the same SQL, sessions serialize within the shared pool
- How to find invalid SQL:
  - Look for parse count failures from v\$sysstat
  - Check session traces for error messages
  - Look for SQL\*Net Break/Reset

# Agenda

- 1 Computer Science Basics
- 2 Schema Types and Database Design
- 3 Database Interface
- 4 DB Deployment and Access Options**
- 5 Application Algorithms
- 6 Resource Management

# Database Deployment And Access Options



# Performance Features

The performance features in the Oracle Database mostly focus on one of two areas

- Reducing the amount of time it takes to get the data
  - Indexes
  - Partitioning
  - Clustering
  - Caching
- Reducing the amount of time it takes to process the data
  - Set processing
  - Caching
  - Bloom filters

# Performance Features

Some of the performance features are implemented Up-Stack

- Up-stack technologies require admins (people) to use DDL to *do* something, resulting in permanent objects, segments, or structures
- Examples: Partitioning, Compression

Some of the performance features are implemented Down-Stack

- Down-stack technologies are more transient and take advantage of metadata gathered during data population or through system use
- Examples: Exadata Storage Indexes, Bloom Filters

# Indexes

- The Good

- Indexes are a way to reduce the amount of IO

1. Lookup a value in the index
2. Get the rowid(s)
3. Retrieve the row(s) from the table

- This technique works well when retrieving a small number of rows

- The Bad

- This is not an efficient way to retrieve a large number of rows

- The index needs to be maintained during any DML activity on the indexed columns

- This is noticeable with a large number of rows

- Can provide more scope for suboptimal execution plans because there are simply more plan choices

# Indexes

## Some basic math

- Index driven query retrieving 1,000,000 rows
  - Assume the index is cache and the data is not.
    - 1,000,000 random IOPS @ 5ms per I/O
    - This required 5000 Seconds ( or over 1 hour ) to Execute
  - How much data could you scan in 5000 Seconds with a fully sized I/O system able to scan 25 GB/Sec ?
    - Over 100 TB !
- Clearly for large Oracle databases the game has changed:
  - DBAs and Data Architects need to design based on the system IO capabilities

# Indexes

## Index Types

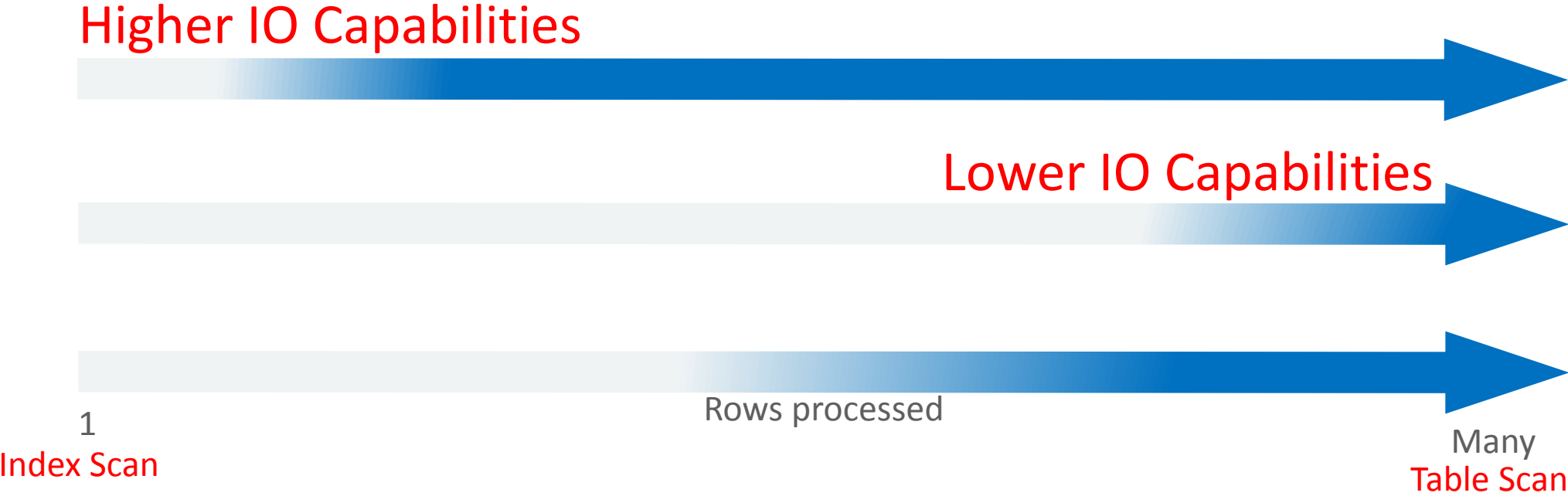
- Btree Indexes
  - Well understood for many years – essential for OLTP operations
  - Challenges
    - Contention
    - Maintenance overhead
    - Big/small rows challenges
- Bit Mapped Indexes
  - Create quickly
  - Compact
  - Challenges
    - Require regular rebuilding if subject to trickle loads
    - Big/small rows challenges

# Table Scans

- Without indexes, the database needs to scan the table to retrieve rows
  - This is inefficient when retrieving a small number of rows
- Table scans are faster when retrieving a large number of rows with sufficient IO bandwidth and CPU resources
  - Exadata
  - DBIM
- There are a number of database features to make table scans more efficient
  - Partitioning
  - Compression
  - Clustering
  - Storage Indexes

# Indexes vs Table Scans

The inflection point for index scans vs table scans is dependent on the IO capabilities of the platform



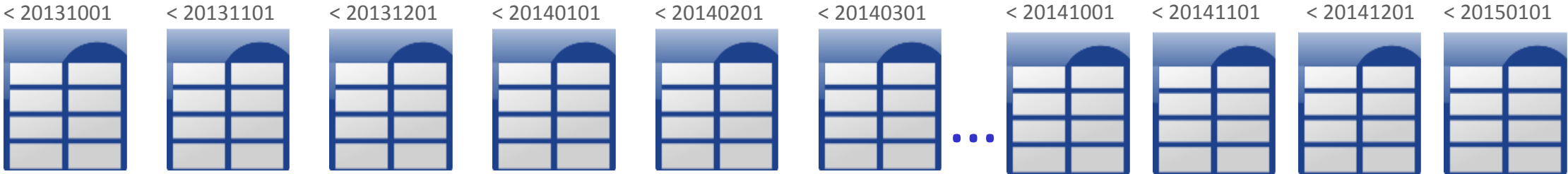
# Partition Design

## Challenges

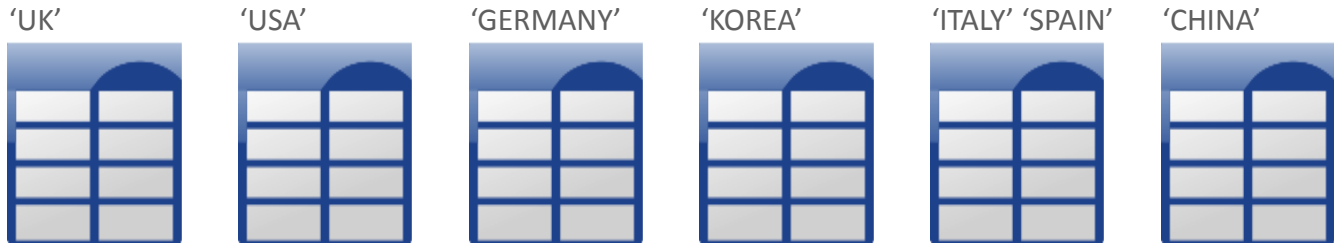
- Partitioning is an important design consideration especially for large databases
- Whenever you partition a table there are positive and negative effects
- The skill of the Database Architect is to devise a partitioning strategy that allows the Database to grow and maintain query performance and data management activities over time.
- In many cases these decisions are made by strength of personality rather than by numerical evidence. Good Architects will prototype various partition designs and evaluate the performance before final selection

# Partitioning Types

- RANGE Partitioning – each partition has upper boundary



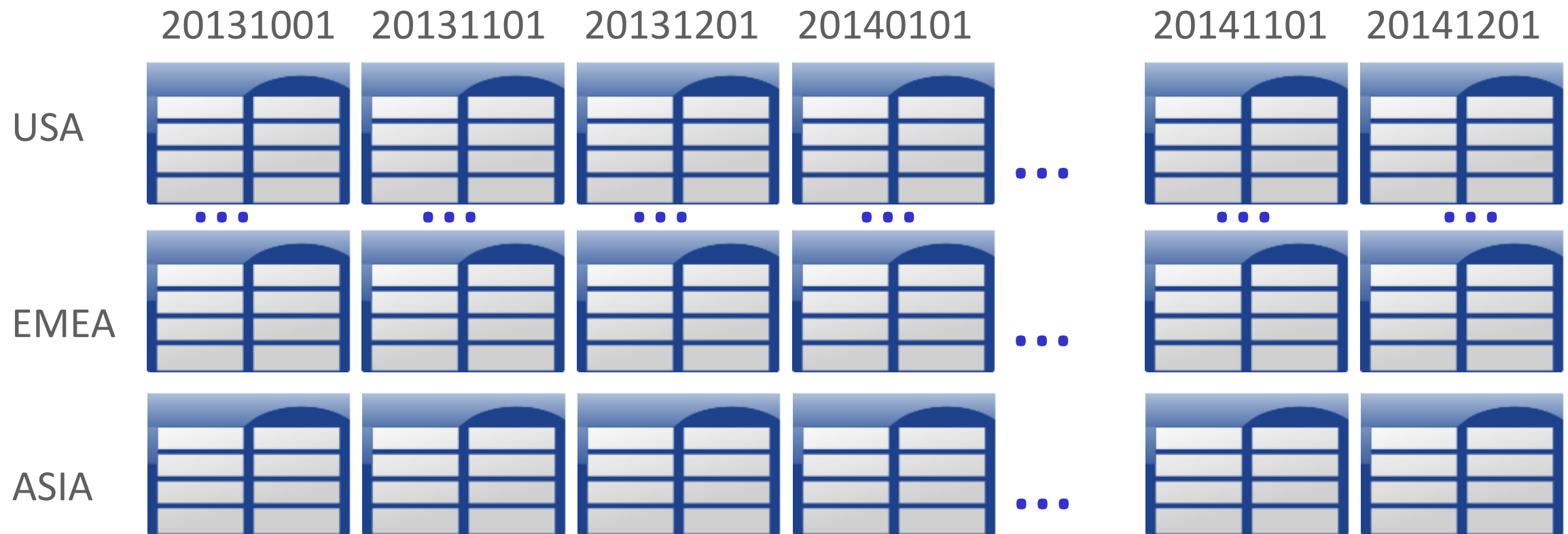
- HASH Partitioning – data is distributed based on a hash distribution of the partitioning key
- LIST Partitioning – each partition contains predefined value(s) of a defined key



# Composite Partitioning

Data is organized along two dimensions

Row placement is deterministically identified by dimensions—RANGE-LIST Example



# Partition Design

## Goals

- Data Management
  - Exchange data in/out
  - Dynamically add and remove partitions
  - Break down operations into smaller pieces

```
ALTER TABLE SALES EXCHANGE PARTITION PART_201409  
WITH TABLE SALES_201409;
```

```
ALTER TABLE SALES MERGE PARTITIONS  
SALES_201310,SALES_201311,SALES_201312  
INTO SALES_2013Q4 COMPRESS;
```

```
ALTER TABLE SALES SPLIT PARTITION SALES_2013Q4  
INTO  
(SALES_201310 VALUES LESS THAN  
'2013-11-01' .....)
```

```
ALTER TABLE DROP PARTITION SALES_2011Q1
```

# Partition Design

## Goals

- Query Optimization
  - Partition pruning
  - Bloom filtering
  - Hash based joins and sorts

# Partition Design

## Common Strategy

- Partition large tables by event date
  - Column chosen should be used in the majority of date based queries
  - Ideally this date will be a non volatile column and will be used as the means to exchange data in/out of the table. If this data is volatile it may imply poor data design
- Subpartition by
  - HASH to support joins/sorts/bloom filtering
  - Another RANGE or LIST to support more pruning

# Partition Design

## Common Mistakes

- DATE RANGE/INTERVAL is either too small or large
  - Remember Exadata does I/O in 1MB requests. With aggressive compression we often see the actual partitions become smaller than 1MB.
- RANGE/LIST strategies that result in partitions with a large variation in size
- The number of HASH Partitions should **ALWAYS** be a power of 2 e.g. 8,16,32,64,128 etc.
  - This is to ensure equally sized partitions
  - Hashing key ought to have lots of values to ensure good distributions – tends to be primary or foreign key
- The number of HASH partitions should
  - Exceed the default DoP of the system by a factor of 2x or more
  - Provide the opportunity for partition-wise joins
- Often partitions are used inappropriately to mitigate index contention – (covered in X-OLTP)
- Too many partitions
  - Oracle support up to 1,048,575 partitions per table or index
  - DML, DDL and metadata operations start to see a noticeable impact on performance around 10,000 partitions

# The effect of too many partitions

- Each partition is an object
- When referenced, an object's metadata needs to be loaded into the shared pool – partitions loaded into the prtmv subpool
- Changing data in a partition may result in the object's stats being marked as stale– involves updating dictionary tables
- Truncating partitions/sub-partitions is a DDL operation – expensive dictionary operation

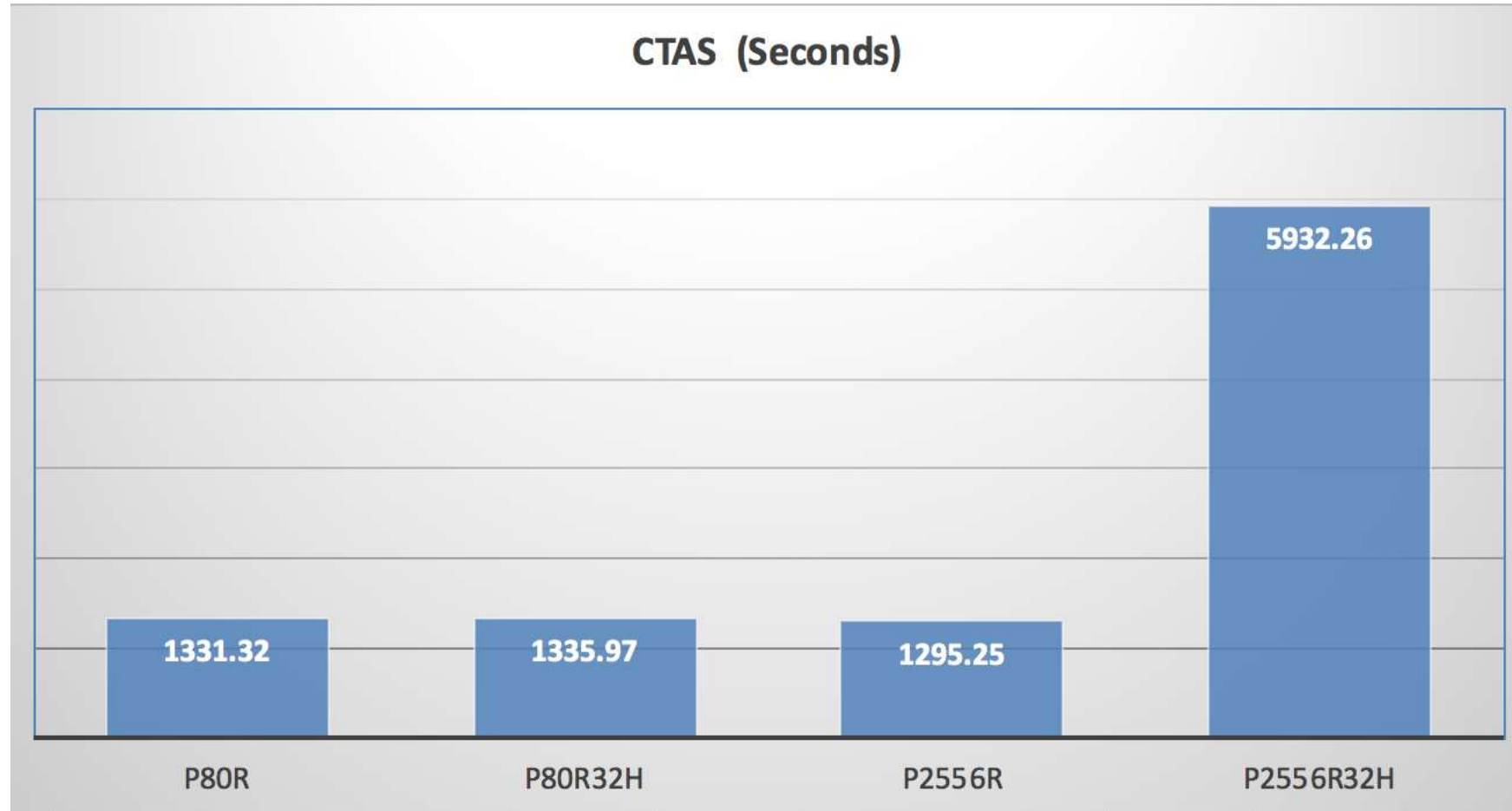
# The effect of too many partitions

## Lab Tests

| Table        | Partitioning            | Subpartitioning        | Total Segments |
|--------------|-------------------------|------------------------|----------------|
| LO_P80R      | Interval/range by month |                        | 80             |
| LO_P80R32H   | Interval/range by month | 32 hash sub-partitions | 2,560          |
| LO_P2556R    | Interval/range by day   |                        | 2,556          |
| LO_P2556R32H | Interval/range by day   | 32 hash sub-partitions | 81,792         |

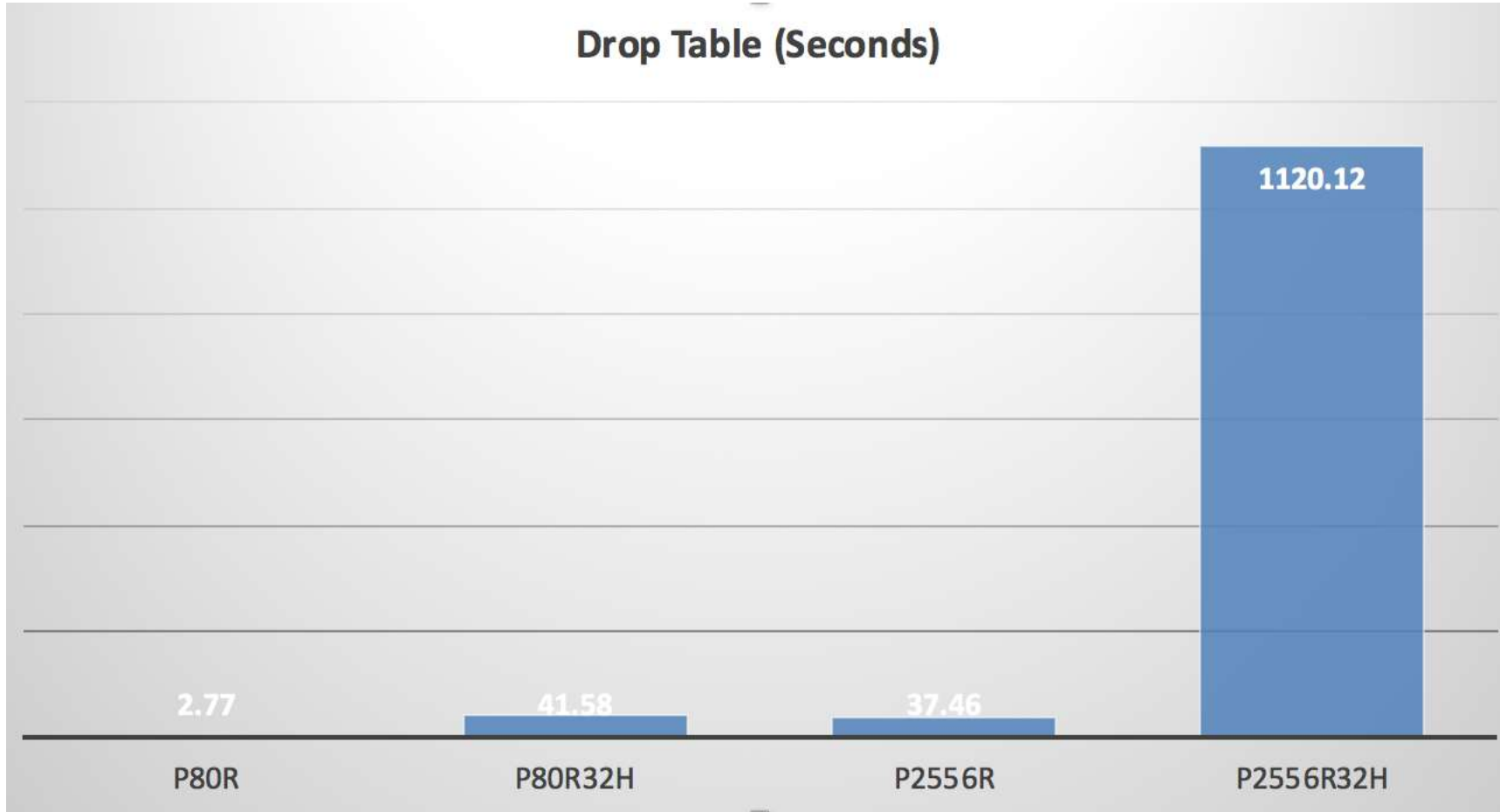
# The effect of too many partitions

## CTAS



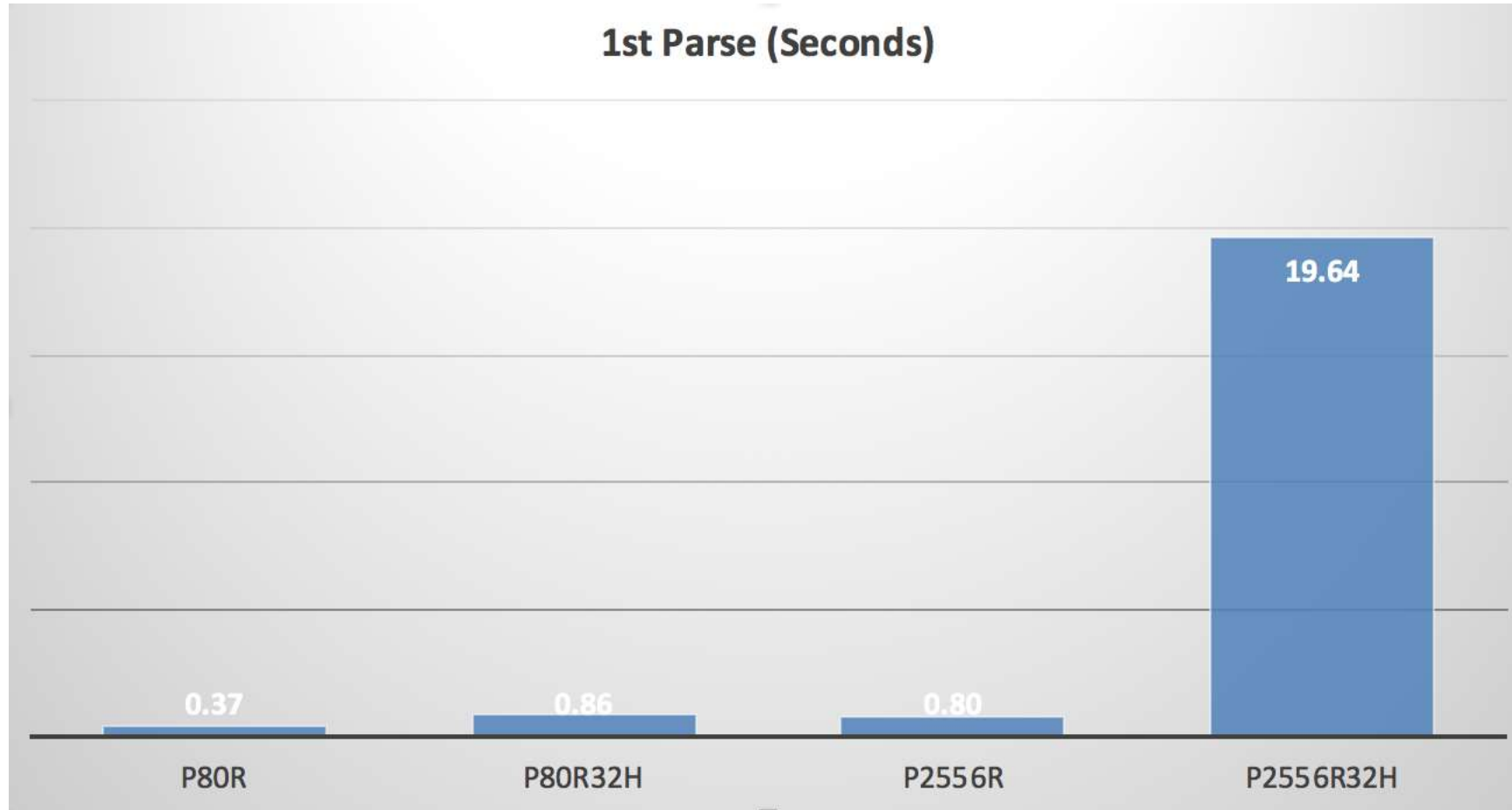
# The effect of too many partitions

## Drop Table



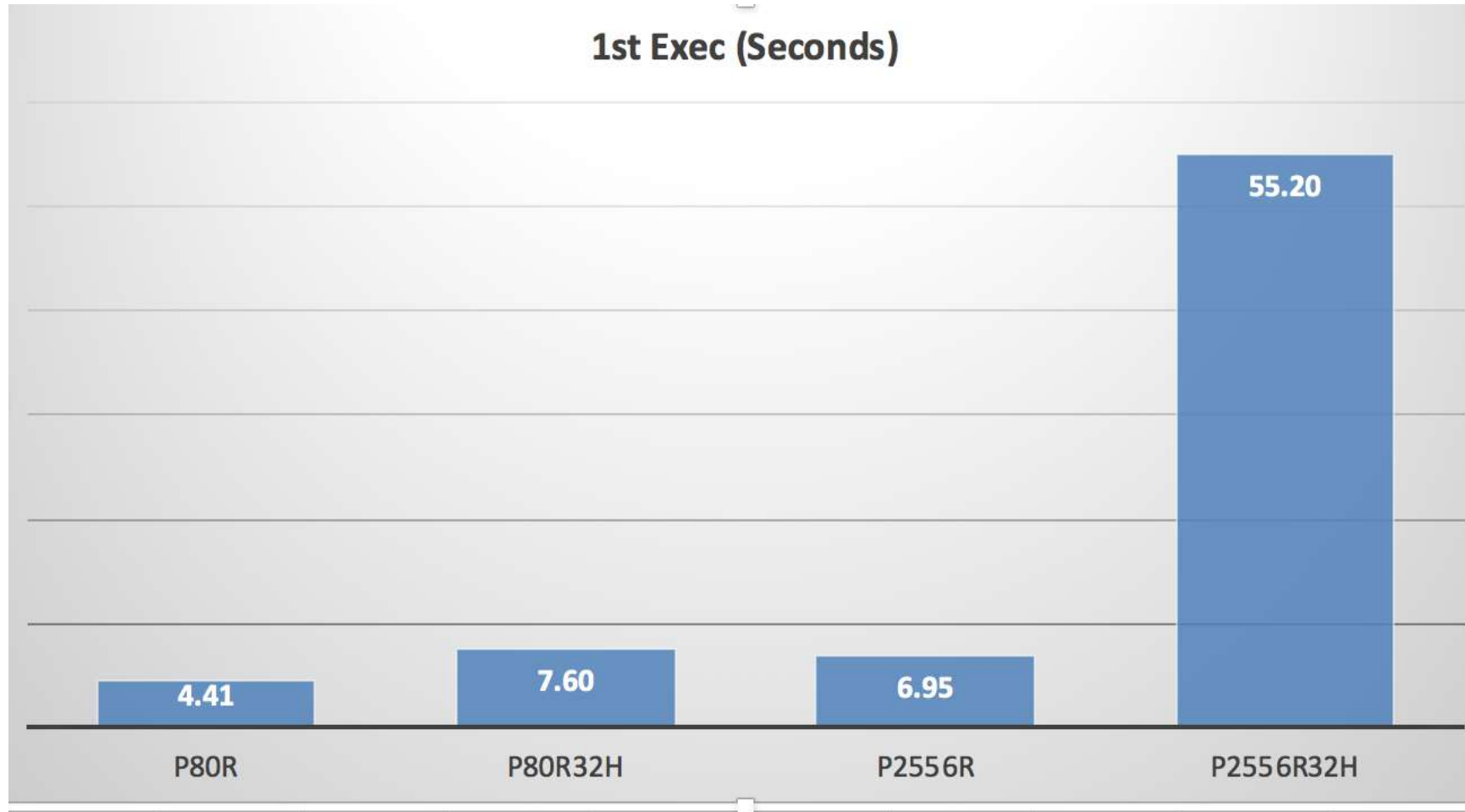
# The effect of too many partitions

## 1<sup>st</sup> Parse



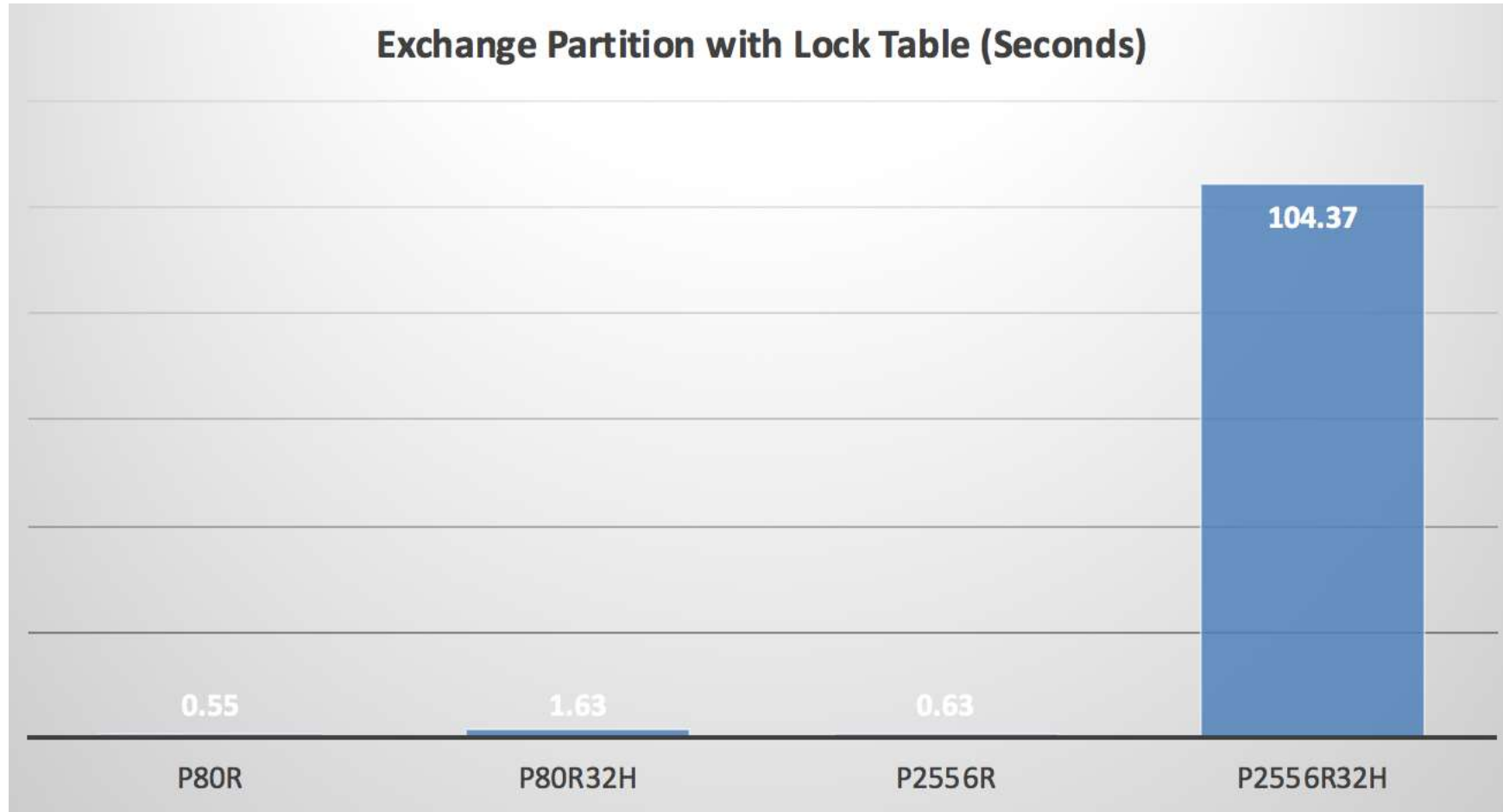
# The effect of too many partitions

1<sup>st</sup> Execute



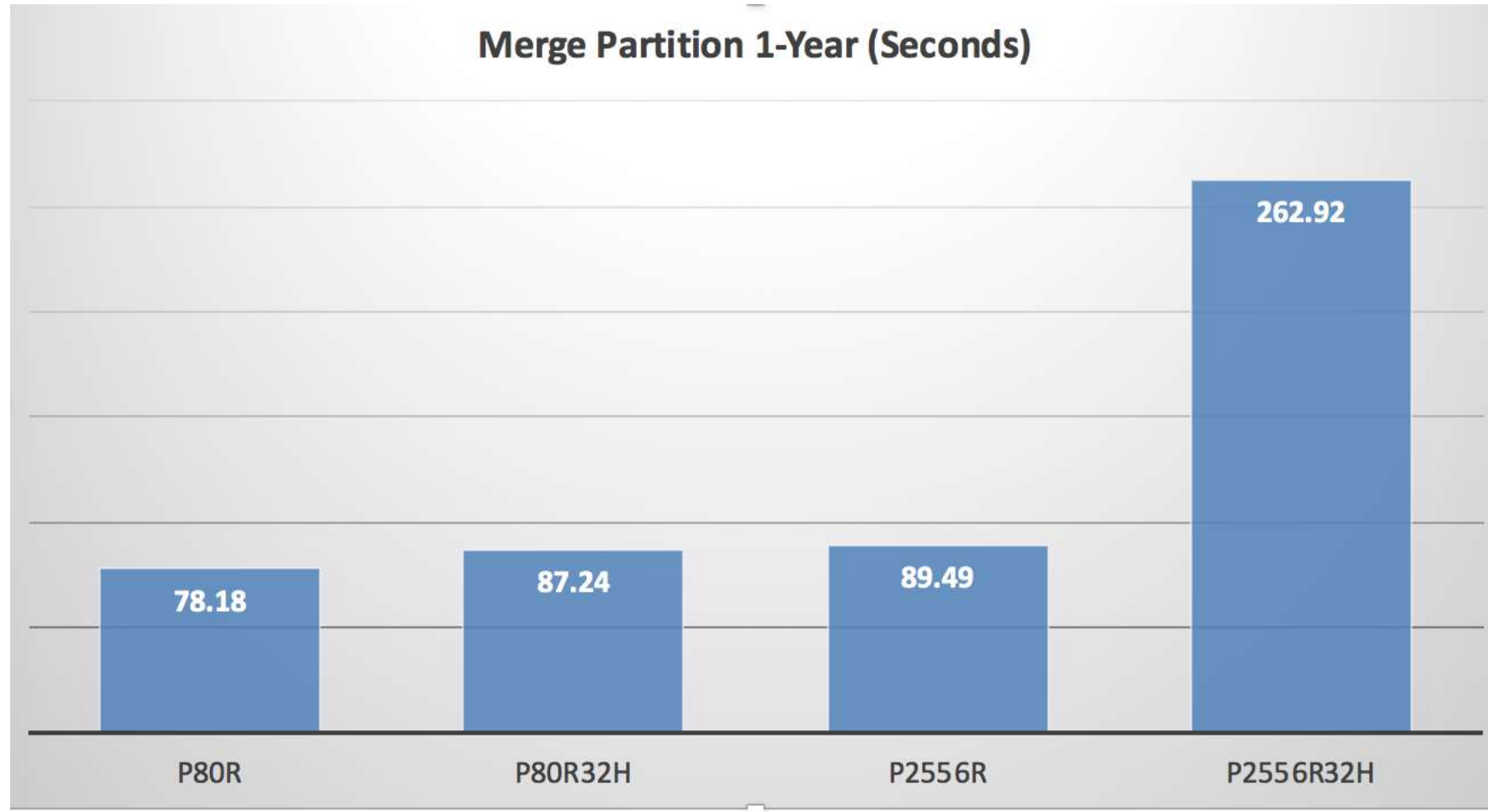
# The effect of too many partitions

## Exchange Partition



# The effect of too many partitions

## Merge Partition



# Customer system with > 9 million active partitions

1. Customer attended RWP training
2. The system processes files and each file is allocated it's own partition
3. After the files are processed, partitions are truncated and returned to the "pool"
4. The next partition allocated is the least recently used – always a new partition added to the shared pool

## Top wait events:

| Event                   | Waits     | Time(s) | Avg wait (ms) | % DB time | Wait Class  |
|-------------------------|-----------|---------|---------------|-----------|-------------|
| library cache lock      | 136,014   | 23,908  | 176           | 26.07     | Concurrency |
| DB CPU                  |           | 18,033  |               | 19.66     |             |
| enq: TM - contention    | 36,084    | 14,074  | 390           | 15.35     | Application |
| read by other session   | 4,725,664 | 9,319   | 2             | 10.16     | User I/O    |
| cursor: pin S wait on X | 79        | 6,020   | 76201         | 6.56      | Concurrency |

# Compression Design

- Compression impacts:
  - Positively by speeding up table scans and reducing disk space/backup requirements
  - Positively by storing more rows in the available buffer cache for OLTP
  - Negatively by slowing down UPDATES, DELETES
- A well designed Database will append only to the biggest tables so high levels of compression would seem appropriate.
  - However not all Databases are well designed and many in fact do revise data over time
  - Here lies the challenge of compression
- Most Oracle compression works on repeated values, random values give no or “negative” compression

# Compression Techniques

| Compression Type       | Description   | Ratio |
|------------------------|---|-------|
| Basic Block based      | Eliminate duplicate values in a database block                                    | 2-4   |
| HCC                    | Compression unit with data organized by column, encoded and compressed            | 6-50  |
| Database In-Memory     | Compression unit with data organized by column, encoded and optionally compressed | 2-50  |
| Flash Compression      | Flash Disk Compression on Exadata Storage Cells                                   | 2     |
| Disk Array Compression | Many Storage Arrays can compress the data before writing to disk                  | N/A   |
| Network Compression    | Can be done by Oracle or specialized network hardware                             | N/A   |

*The numbers above are based on real-world experience. Generated data may give widely varying results.*

# Clustering

- Mechanisms of co-locating data with common values within a table
- If you co-locate data you can retrieve them with the minimal amount of IO
- Sometimes data is naturally clustered due to the method of loading (eg transaction data loaded by day will naturally be grouped by that day)
- Other methods can be used to guarantee the co-location of the data

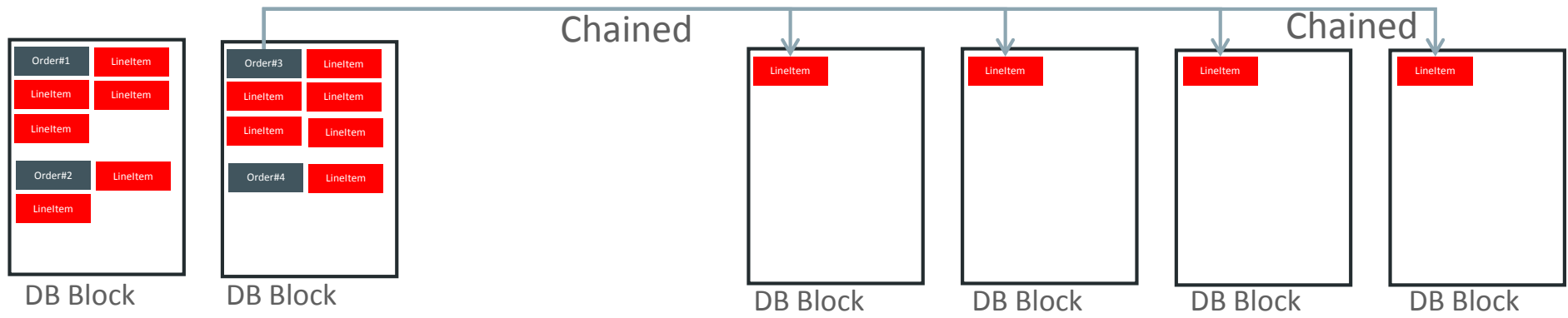
# Techniques used for clustering

- Sorting the data before loading means that the data will be naturally sorted around the sort key, reducing the number of physical I/O's required to retrieve rows with the predicate based on the sort key if retrieving data via indexes or storage indexes
- Clusters
  - Groups of one or more tables that are clustered around a common key which can be hashed to determine position
  - If we access rows on the key then we can retrieve all the related data

# Clusters

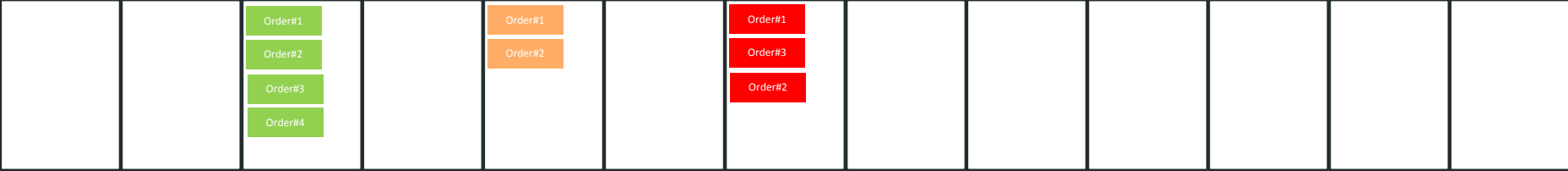
## B\*Tree Clusters

- Single or Multiple tables physically clustered around a defined clustering key; access to cluster key via index



## Hash Clusters

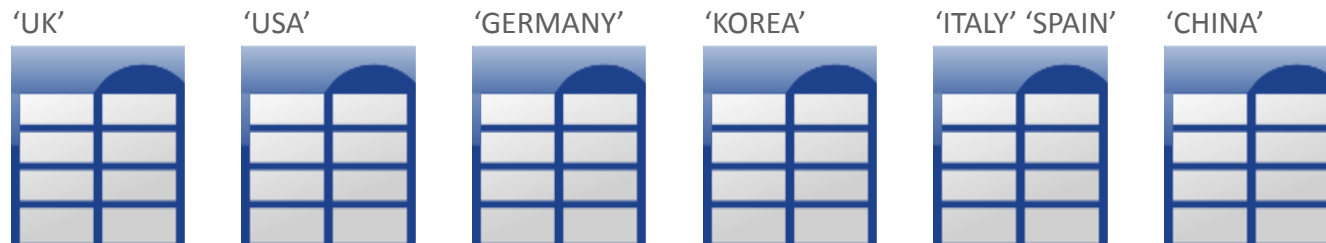
- Tables whose position in the object is determined by a hash function – single IO retrieval based on clustering key, allowing related rows from multiple table to be retrieved quickly:



# Techniques used for clustering

## Partitioning

- Divide data based on the partitioning column
- Only need to access required partitions when data is queried by the partitioning key
- For Example List Partitioning based on country, all the orders for a country would be in a single partition – might lead to asymmetric sizes though



# Techniques used for clustering

## Attribute Clustering—New in 12.1.0.2

- Sorts the data as it is loaded or reorganised on the clustering key
- The rows with a given value of the clustering column are stored using a small number of blocks
- Index retrieval based on clustering columns will perform less IO
- Used in conjunction with Zone Maps to reduce the amount of IO needed to scan a table while searching for clustering column values
- Will improve the efficiency of DBIM queries on the clustering column

# Attribute Clustering

Linear-Ordered Table

| Category | Country |
|----------|---------|
| Boys     | AR      |
| Boys     | JP      |
| Boys     | SA      |
| Boys     | US      |
| Girls    | AR      |
| Girls    | JP      |
| Girls    | SA      |
| Girls    | US      |
| Men      | AR      |
| Men      | JP      |
| Men      | SA      |
| Men      | US      |
| Women    | AR      |
| Women    | JP      |
| Women    | SA      |
| Women    | US      |

Interleaved-Ordered Table

|          |       | Country  |          |          |          |
|----------|-------|----------|----------|----------|----------|
| Category | Women | AR Women | JP Women | SA Women | US Women |
|          | Men   | AR Men   | JP Men   | SA Men   | US Men   |
|          | Girls | AR Girls | JP Girls | SA Girls | US Girls |
|          | Boys  | AR Boys  | JP Boys  | SA Boys  | US Boys  |

# Caching

Caching is used to

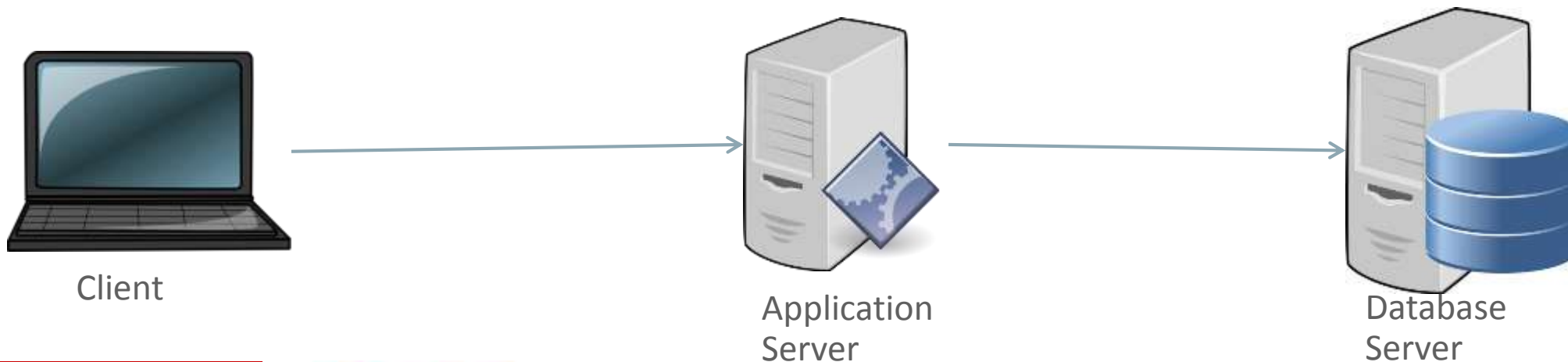
- Reduce the latency of retrieving data
- Reduce or eliminate the work required to process the data

# Caching – what could I cache?

- Data
- Query Results
- Objects

# Caching – which tier would I cache it in?

- Where it is going to be most used
- Outside the database – potential issues if the data changes
  - Client Side – clients do not have the roundtrip to use the data
  - Application Tier – applications can access it directly
- Database Tier



# Database Caching – where would I cache it?



CPU  
1 nano sec



If all data for an operation can be cached

DRAM  
1 micro sec



SGA/PGA, avoids going to IO

Solid State Disk/Flash  
0.01 milli sec



Exadata Flash Cache, Disk Array Solid State Cache

Physical Disk  
1-10 milli sec



# Caching Technologies

## Outside the database

- Client Side Cache
  - Data can be cached on the browser, client or in the application server, fastest access, how is it kept up to date?
- Coherence
  - Application Server Tier Object Cache
- TimesTen
  - Application Server Tier SQL Cache

# Caching Technologies

## Inside the database

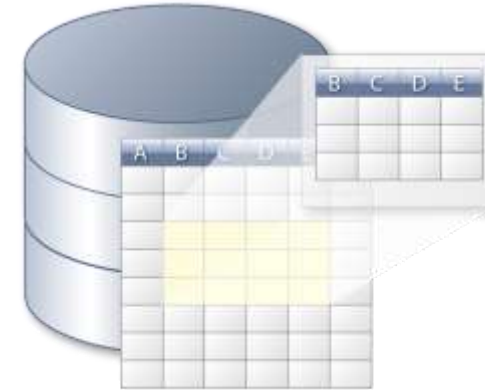
- Buffer Cache
  - Caches frequently used blocks in Memory
  - Avoid disk IO
- Database Smart Cache



# Caching Technologies

## Inside the database

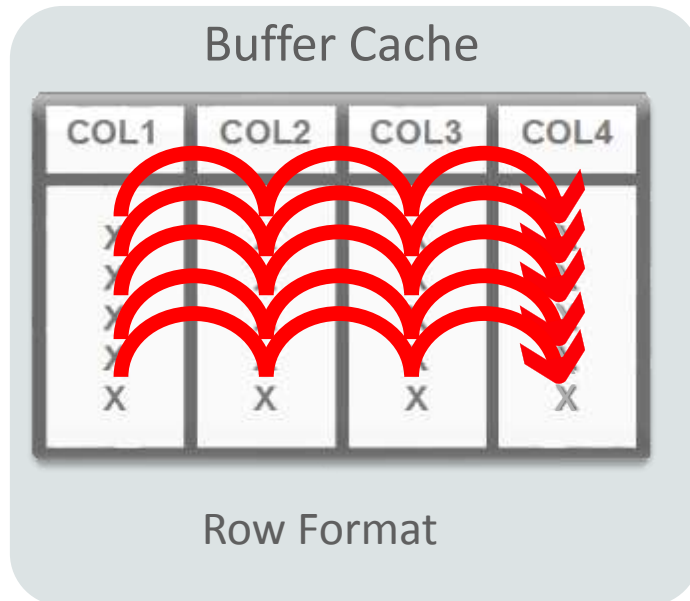
- SQL Result Set Cache
  - Cache queries and sub queries
  - Shared by all sessions
  - Full consistency and proper semantics
- PL/SQL Function Cache
  - Automatic invalidation
  - Shared by all sessions
  - Use for deterministic functions
- Materialized Views
  - Cache summary data which is stored on disk and loaded into the buffer cache.



```
create or replace function cust_count
(p_cust_id number)
return number
result_cache relies_on (orders)
parallel_enable
as
  v_cnt number;
begin
  select count(1) into v_cnt
  from orders
  where customer_id=p_cust_id;
  return v_cnt;
end;
```

# Caching in the buffer cache

- We can evaluate millions rows/sec per core when scanning through the buffer cache.
- For large datasets, restricted by the row format
- 1000's of expensive instructions per column evaluation



```
SELECT COL4 FROM MYTABLE  
WHERE COL3 in ('1', '22', '55');
```

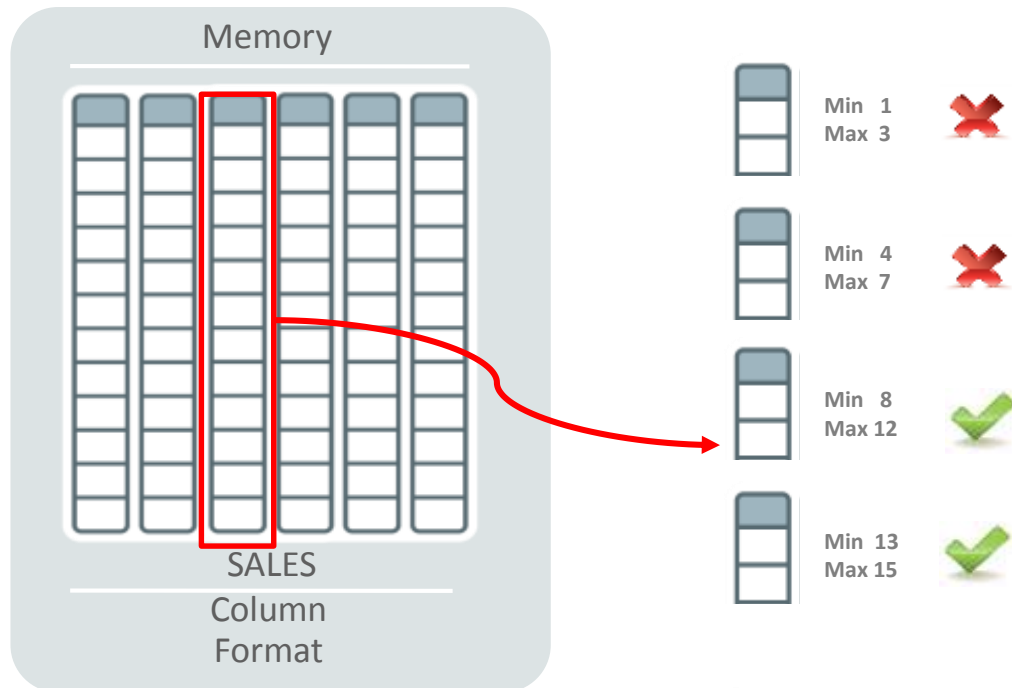


**RESULT**

- Many CPU instructions
- Poor CPU efficiency; extra data and instructions required

# Columnar Format

**Example:** Find sales from stores with a store\_id of 8 or higher



- Each column is transformed to multiple in-memory column units
- Predicates are evaluated by interrogating the appropriate column store, 1000's of evaluations can be done with few instructions.
- Min / max value is recorded for each column unit in a storage index
- Evaluate billions column values/sec rows/sec/core

- Few CPU instructions
- High CPU efficiency; pipeline only columns we need to CPU & use SIMD

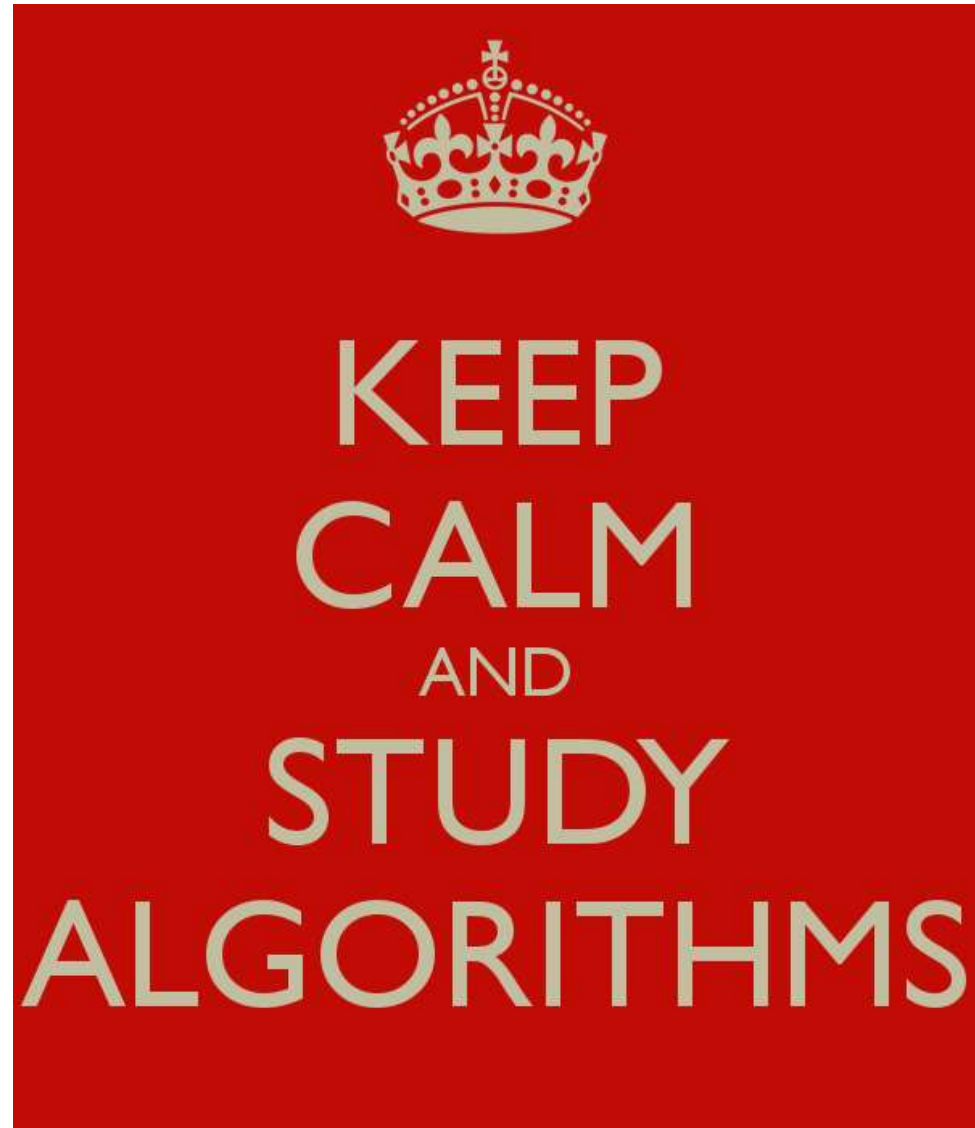
# Database In-Memory Basics

- In-memory is best at evaluating large numbers of rows with few column predicates and projecting few columns
- Queries must perform FULL table scans
- Does not help index access methods
  - Are indexes being used inappropriately ?
    - Few rows
    - Many rows
- Does not help queries bound by sort/aggregate
- **Know where your problem is!**

# Agenda

- 1 Computer Science Basics
- 2 Schema Types and Database Design
- 3 Database Interface
- 4 DB Deployment and Access Options
- 5 Application Algorithms
- 6 Resource Management

# Application Algorithms



# Data Processing Techniques

# Small Numbers and Large Multipliers

How long does it take when processing one row per millisecond?

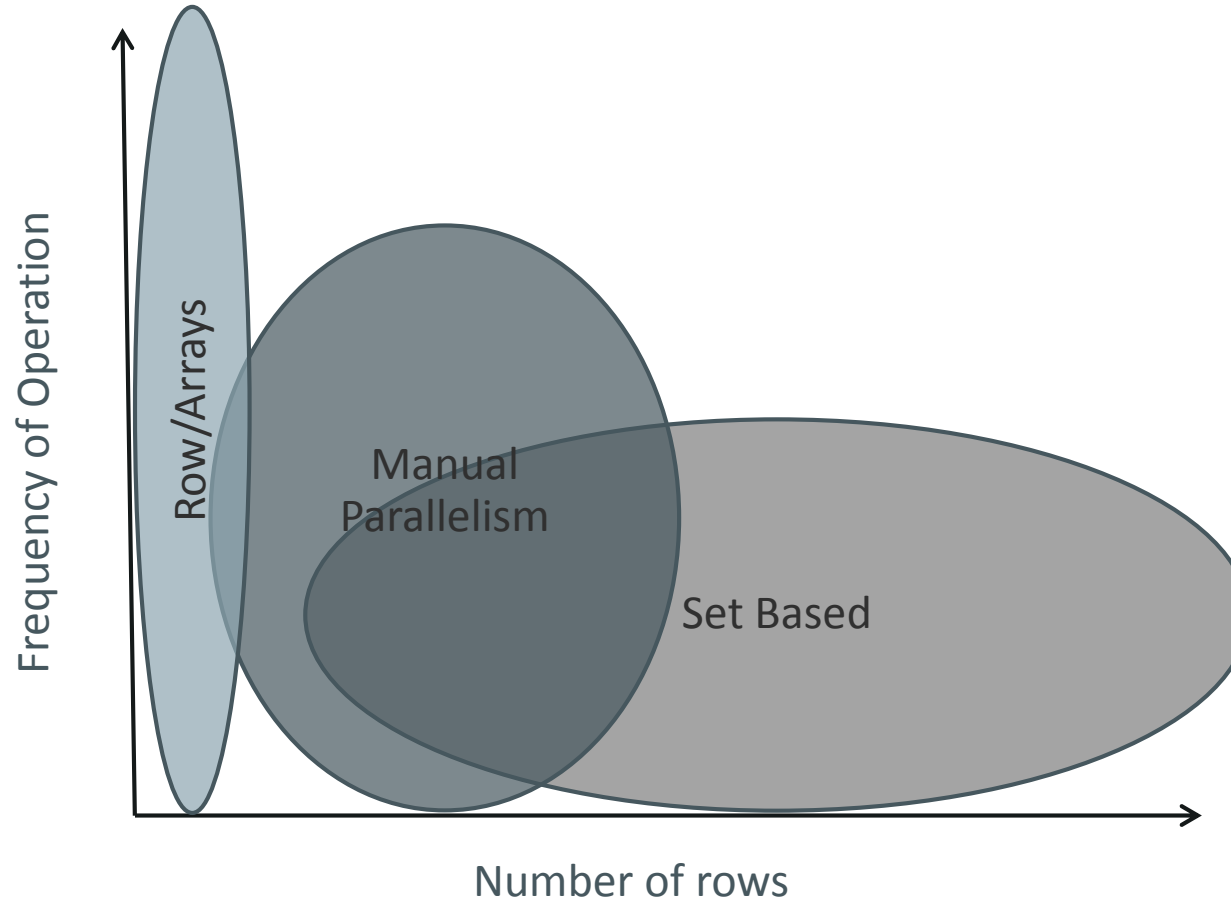
| Table    | Elapsed Time in Seconds | Wall Clock Time |
|----------|-------------------------|-----------------|
| Thousand | 1                       | 1 second        |
| Million  | 1,000                   | 17 minutes      |
| Billion  | 1,000,000               | 12 days         |
| Trillion | 1,000,000,000           | 32 years        |

# Data Processing Techniques

- Row by row
  - Procedural processing one row at a time
- Arrays
  - Procedural processing of a set of rows at a time
- Manual Parallelism
  - Multiple threads/processes each performing row by row or array processing
- Set based processing
  - Process data in groups or sets of rows

# Application Algorithms

## Optimal Usage



# Row by Row



# Data Processing Techniques

- Row by row
  - Procedural programming
  - Single process loops through and processes data one row at a time
  - Good for operations on small data sets
  - Often data is retrieved from the database, processed in the middle tier and then inserts, updates or deletes are performed in the database
  - The processing time can be very long for large data sets
    - To process 1 million rows at 1 ms per row takes 1,000 seconds or 17 minutes

# Data Processing Techniques

## Row by row

```
declare
  cursor c is select s.* from emp s;
  r c%rowtype;
begin
  open c;
  loop
    fetch c into r;
    exit when c%notfound;
    if r.deptno = 20 then
      insert into west d values r;
    else
      insert into east d values r;
    end if;
    commit;
  end loop;
  close c;
end;
/
```

# Arrays



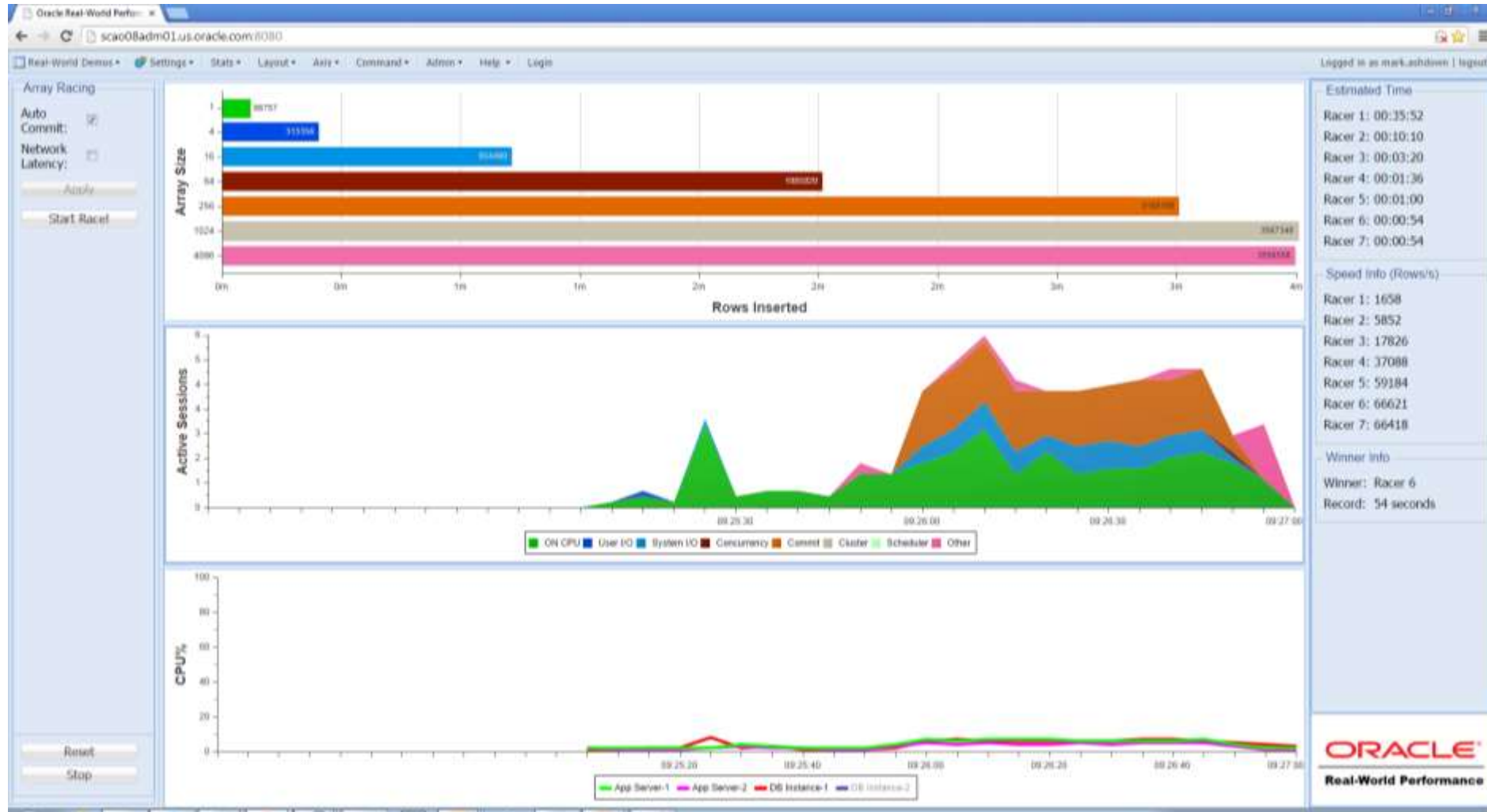
# Benefits of Arrays

- Usually fairly simple to change a row by row process to use an array
- Arrays allow the database to process a set of rows at a time
  - Reduces network round trips
  - Reduces COMMIT time
  - Reduces code path in client and server
- Making a single process faster due to improved efficiency

# Array Racing Demo



# Array Racing With Auto Commit



# Array Racing With No Auto Commit



# Manual Parallelism



# Data Processing Techniques

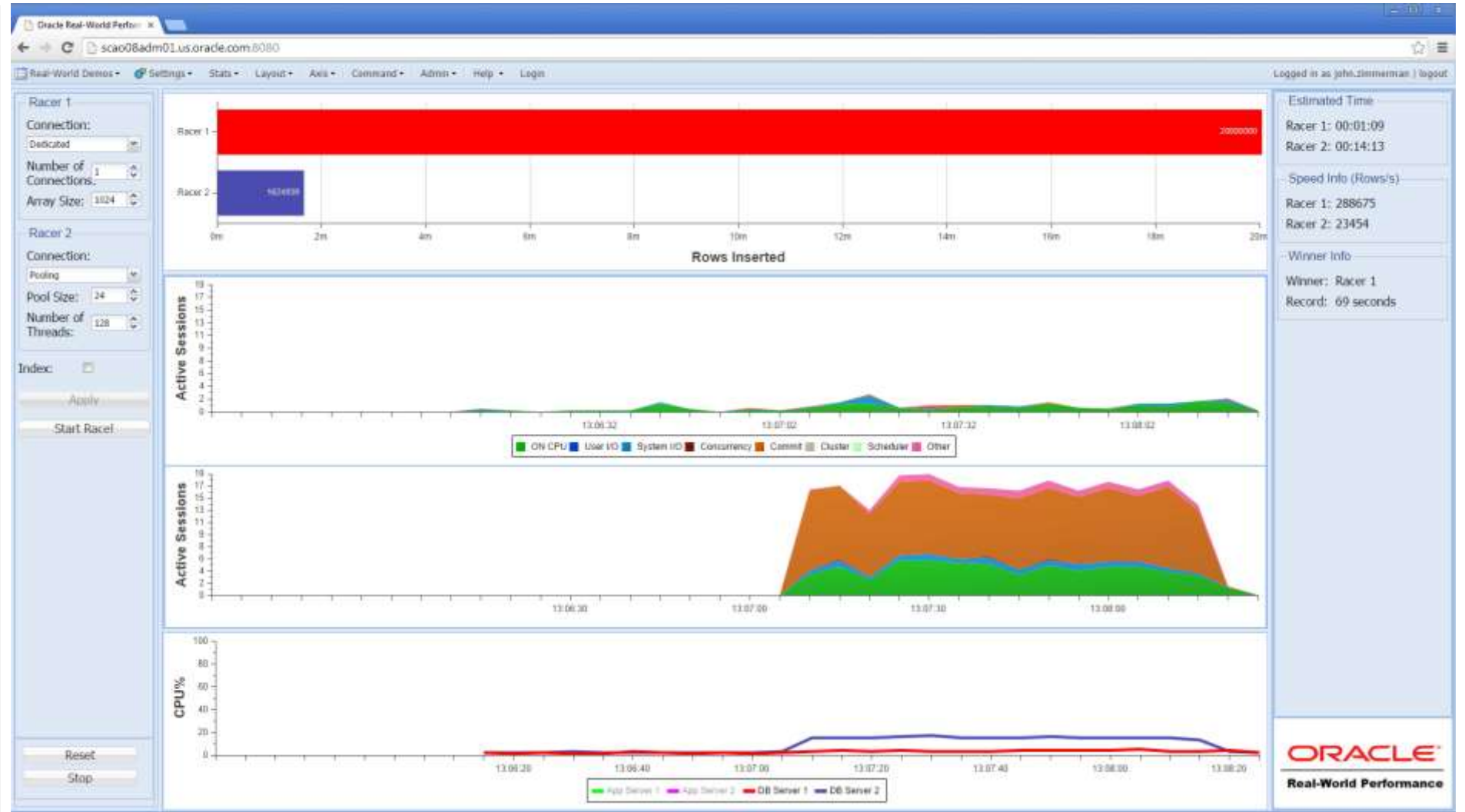
- Manual Parallelism
  - Parallelization of the row by row technique
  - Multiple threads/processes each performing row by row or array processing
  - Often controlled by a middle tier or ETL server
  - Complex to implement and evenly distribute the workload over many threads
  - Possible contention issues with multiple threads performing the same operations on database objects

# Array vs Manual Parallelism Demo



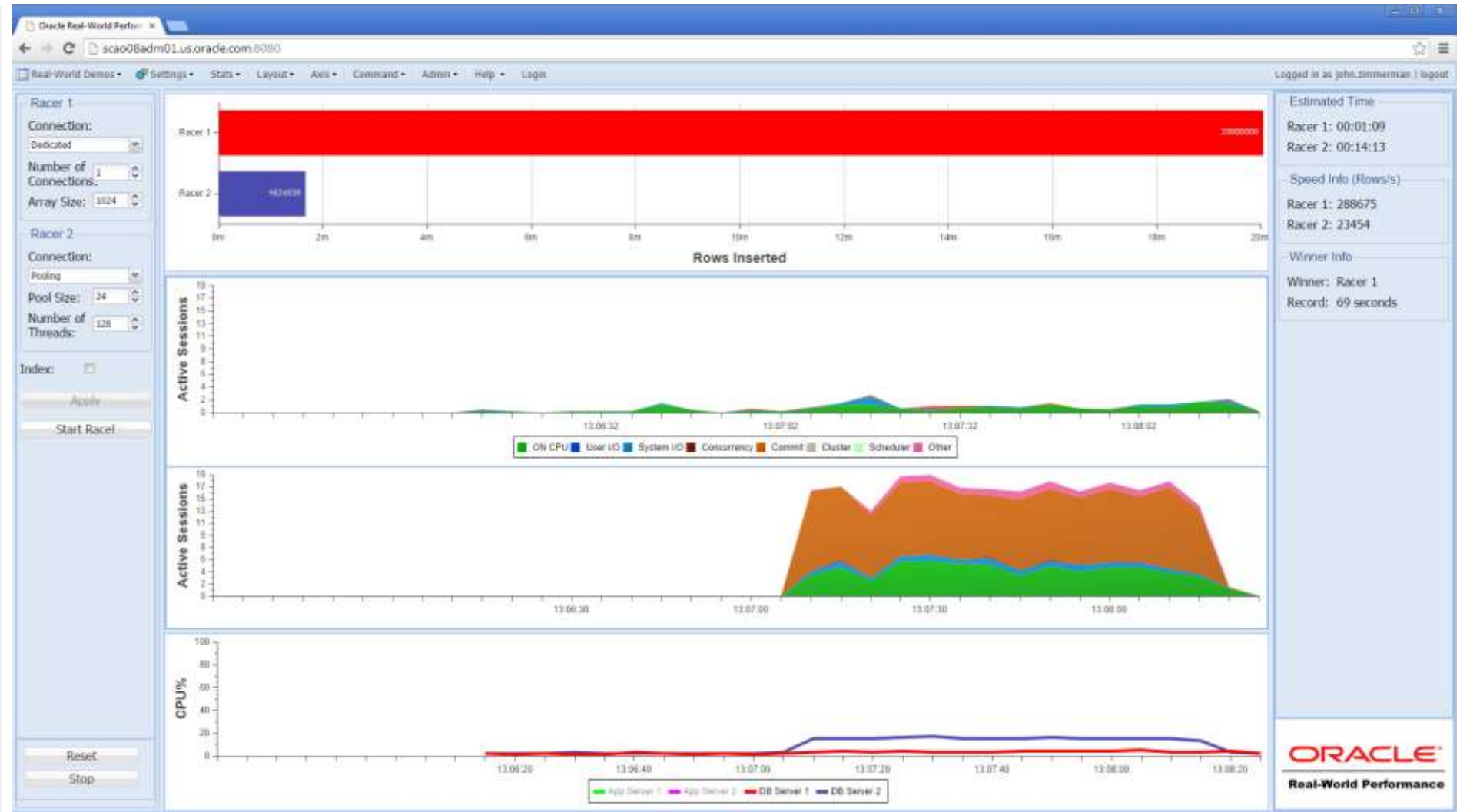
# Arrays vs Manual Parallelism Scenario

- Already have a service that processes a single message
- Process more messages concurrently by using multiple threads or change the service to process a set of messages.
- Which is better?



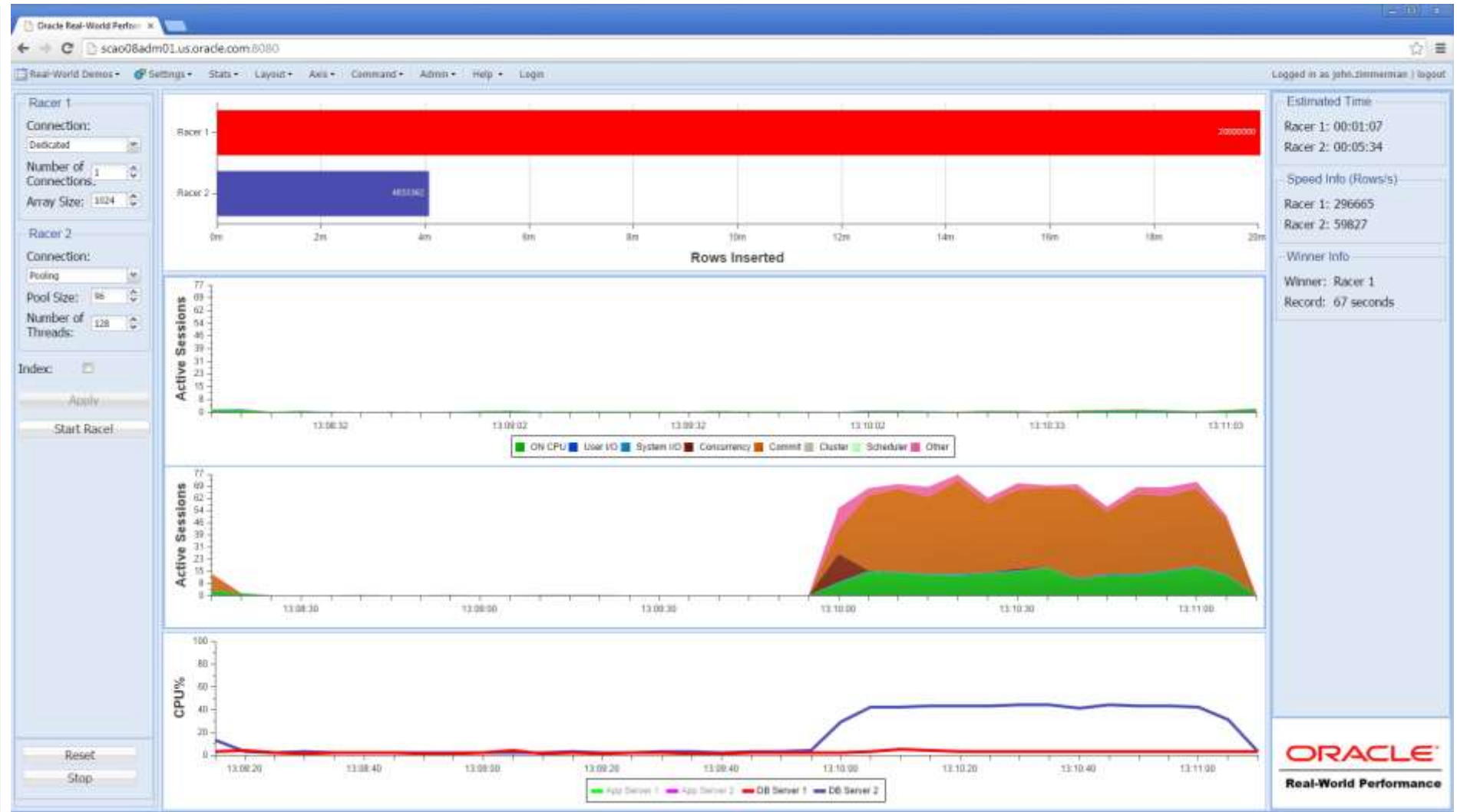
# Arrays vs Manual Parallelism Analysis

- Not much of a contest
- Array takes 1minute
- Threads takes 14 minutes
- With higher load on server
- But still resource available
- Larger pool size for threads?



# Arrays vs Manual Parallelism Analysis

- 4x threads improved performance
- Array takes 1minute
- Threads takes 5 minutes
- Much higher load on server
- This is ONE dedicated process vs 64 parallel threads



# Arrays vs Manual Parallelism Scenario

- What happens when we add an index to both systems?



# Arrays vs Manual Parallelism Analysis

- Both systems slowed by index
- Multiple threads suffers from contention, buffer busy wait, TX index contention



# Set Based Processing



# Data Processing Techniques

- Set based processing
  - Process data in groups or sets of rows
  - Use SQL to define the results
  - Database processes the data efficiently in sets
    - No movement of data over the network
    - Hash joins
    - Parallel Query and DML
    - CPU and IO capabilities exploited with Engineered Systems

# Data Processing Techniques

## Set based processing

```
insert /*+ append */ into west
select *
from emp
where deptno = 20;

commit;
```

```
insert /*+ append */ into east
select *
from emp
where deptno != 20;

commit;
```

# Data Processing Techniques

## Multiple ways to get the same result

```
insert /*+ append */ first
  when deptno = 20 then
    into west values ...
  else
    into east values ...
select *
from   emp;

commit;
```

# Set based processing

- Use SQL to define the results
- Let the DB figure out how to do it
  - Some SQL operations:
    - create table as select
    - insert /\*+ append \*/ select
    - intersect
    - minus
    - exists
    - not exists
    - window functions
    - multi-table inserts
    - outer joins

# ETL DML Challenges

## Real World Debate

- The performance gains are often 2-3 orders of magnitude
- The challenge is re-writing code to exploit these techniques
  - Often emotional rather than technical
- There is an OLTP vs DW culture

# Set Programming: Loading

**ORACLE®**  
RWP Video



**ORACLE®**

**ORACLE®**  
REAL-WORLD PERFORMANCE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved. |

# The Example Code

## Why PL/SQL?

- PL/SQL is a great way to submit SQL to the database
  - Many features for managing the execution of SQL
  - Eliminates network overheads
  - Optimizes commit time
- All the examples use PL/SQL
- When the sample code runs slowly, all the time is in SQL
  - The application algorithm is sub-optimal and NOT
    - PL/SQL
    - SQL

# The Example Code

- The example code demonstrates the different techniques
- The example code is not intended to be best practice!
  - No exception handling
  - No restart capability

# Code Conventions

| Table Alias | Usage             |
|-------------|-------------------|
| s           | Source table      |
| d           | Destination table |

# Code Conventions

| Variable Name | Usage   |
|---------------|---|
| a             | Associative array (formerly index-by table) for rows read from or written to the database           |
| c             | Cursor for rows read from the database  |
| i             | Generic loop index  |
| r             | Record for row read from or written to the database   |
| t             | Type definition for an associative array (formerly index-by table), usually based on a source table |

# Loading using Row-by-Row Method

```
declare
```

```
cursor c is select s.* from ext_scan_events s;
```

```
  r c%rowtype;
```

```
begin
```

```
  open c;
```

```
  loop
```

```
    fetch c into r;
```

```
    exit when c%notfound;
```

```
    insert into stage1_scan_events d values r;
```

```
    commit;
```

```
  end loop;
```

```
  close c;
```

```
end;
```

# Loading using Array Method

```
declare
```

```
cursor c is select * from ext_scan_events;
```

```
type t is table of c%rowtype index by binary_integer;
```

```
a t;
```

```
rows binary_integer := 0;
```

```
begin
```

```
open c;
```

```
loop
```

```
fetch c bulk collect into a limit array_size;
```

```
exit when a.count = 0;
```

```
forall i in 1..a.count
```

```
insert into stage1_scan_events values a(i);
```

```
commit;
```

```
end loop;
```

```
close c;
```

```
end;
```

# Loading using Home-grown Method

```
declare
  sqlstmt varchar2(1024) := q'[
-- BEGIN embedded anonymous block
cursor c is select
      s.*
from
      ext_scan_events_${thr} s;
type t is table of c%rowtype
  index by binary_integer;
a t;
rows binary_integer := 0;
begin
for r in (select
      ext_file_name
from
      ext_scan_events_dets
      where ora_hash(file_seq_nbr,${thrs}) = ${thr})
loop
...

```

# Loading using Home-grown Method

```
...
loop
  execute immediate
    'alter table ext_scan_events_${thr} location ' ||
    '(' || r.ext_file_name || ')';
  open c;
  loop
    fetch c bulk collect into a limit ${array_size};
    exit when a.count = 0;
    forall i in 1..a.count
      insert into stage1_scan_events d values a(i);
    commit;
  end loop;
  close c;
end loop;
end;
-- END    embedded anonymous block
]';
...

```

# Loading using Home-grown Method

...

begin

```
sqlstmt := replace(sqlstmt, '${array_size}'  
                  , to_char(array_size));  
sqlstmt := replace(sqlstmt, '${thr}', thr);  
sqlstmt := replace(sqlstmt, '${thrs}', thrs);  
execute immediate sqlstmt;
```

end;

# Loading using Set-based Method

```
alter session enable parallel dml;
```

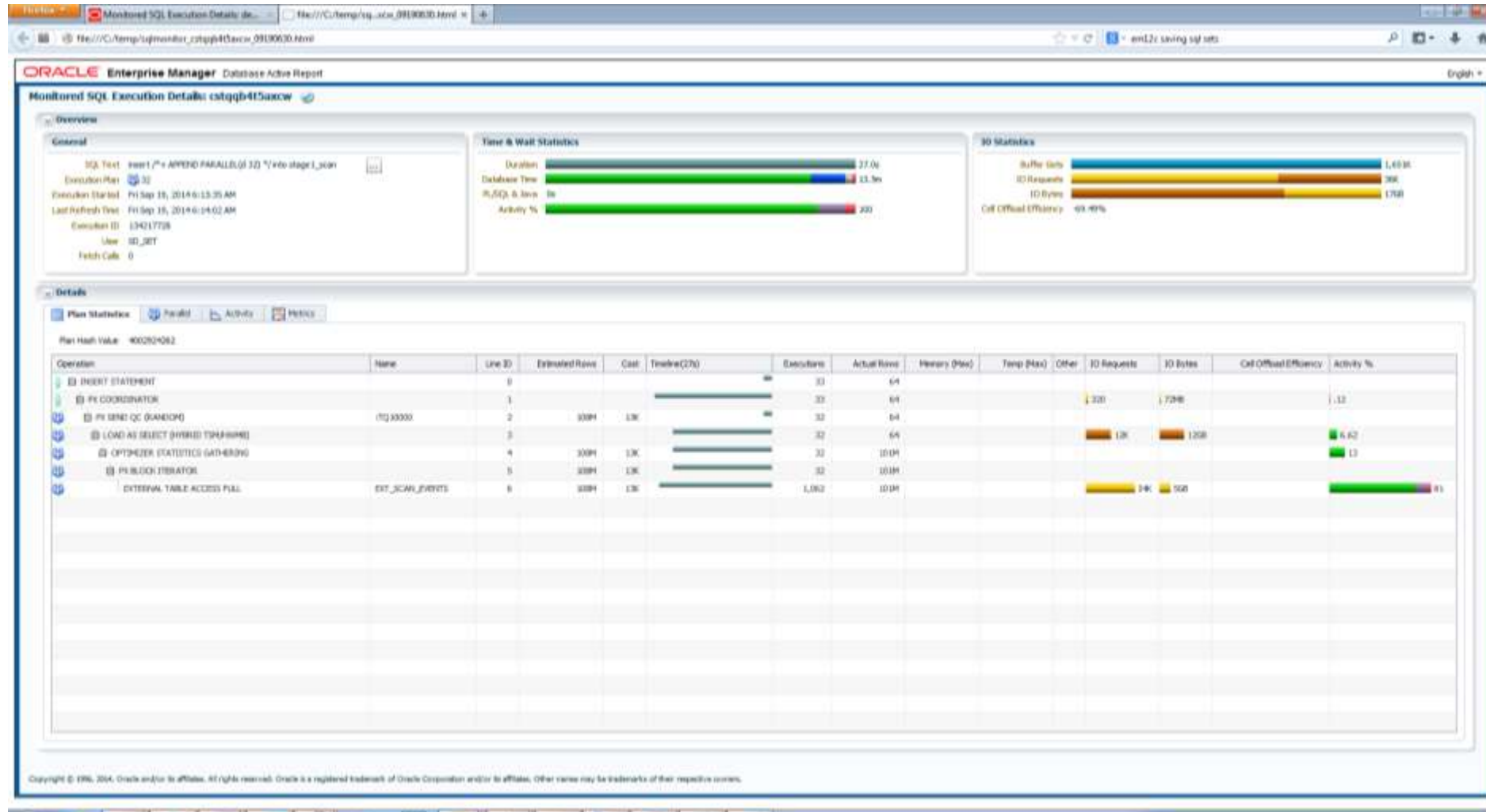
```
insert /*+ APPEND */ into  
  stage1_scan_events d  
select  
  s.*  
from  
  ext_scan_events s;
```

```
commit;
```

# Loading

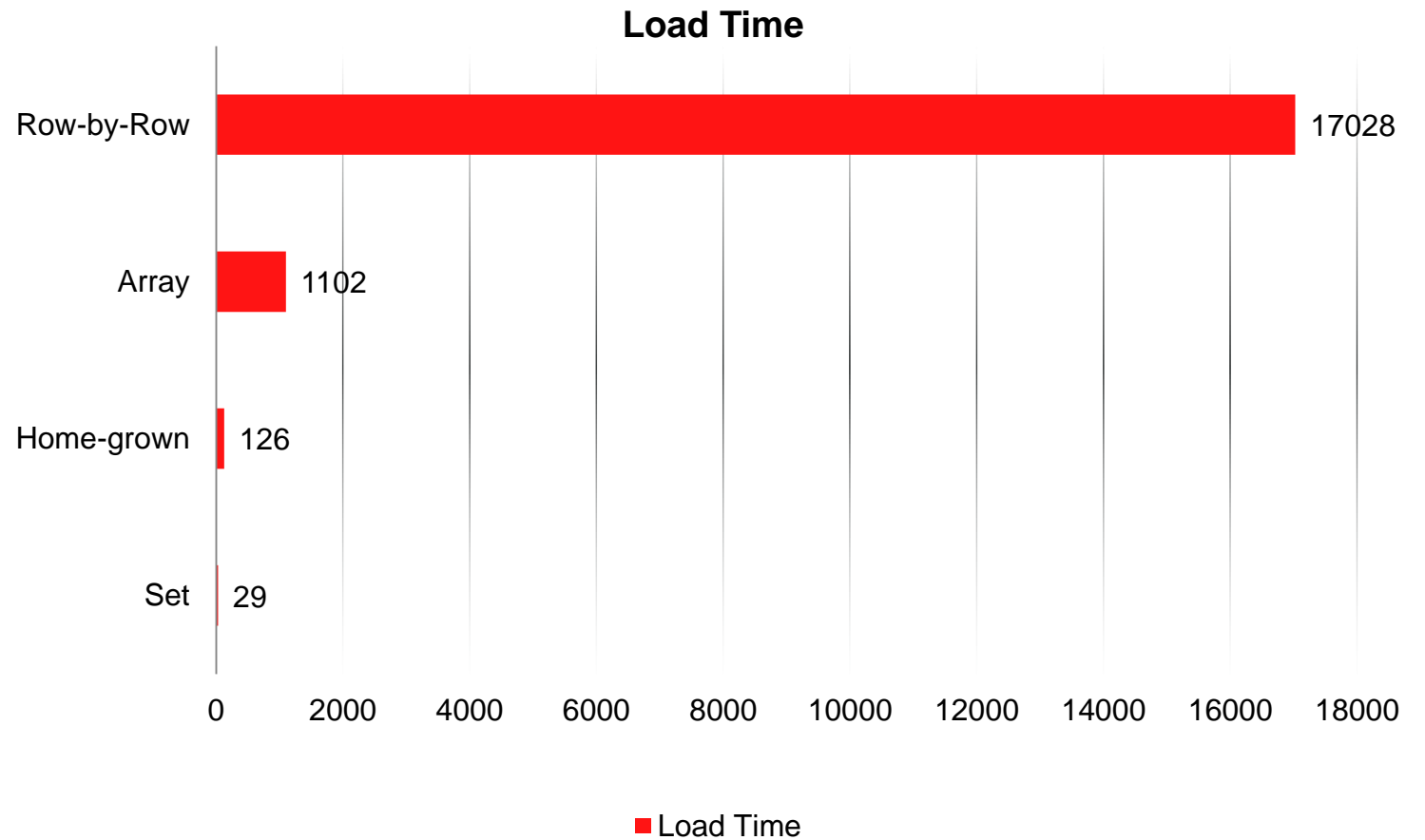


# Loading



# Loading

- Load time in seconds for 100M raw scan events in external files
  - Array size 100
  - Jobs 32
  - Degree of Parallelism 32



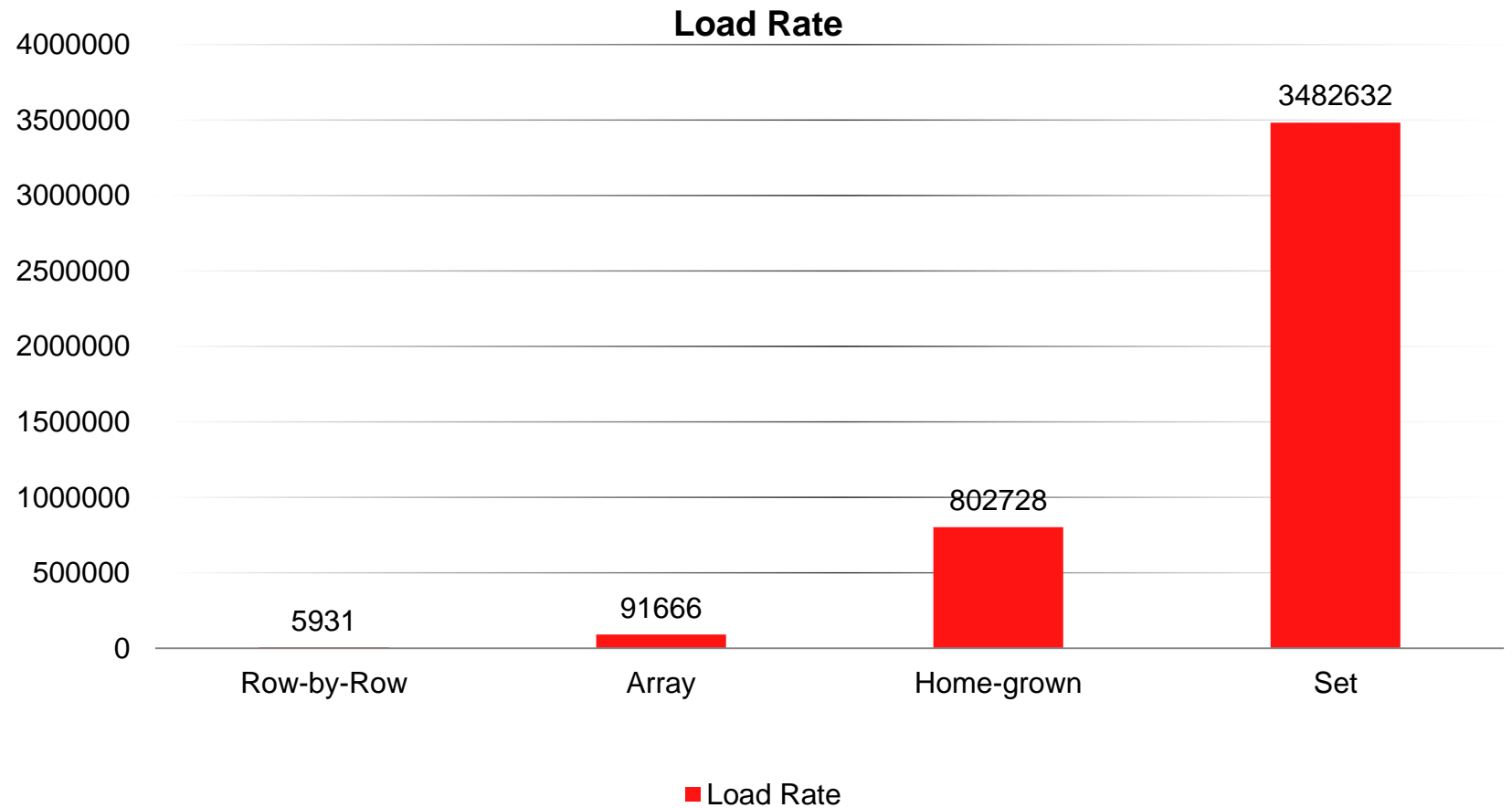
# Loading

Load rate in rows per second  
for 100M raw scan events in  
external files

Array size 100

Jobs 32

Degree of Parallelism 32



# Set Programming: De-duplicating

**ORACLE**  
RWP Video



# De-duplicating using Row-by-Row Method

```
declare
  cursor c is select
    s.*
    ,cast(null as number(10)) as error_ind
    ,rowid as rid
  from
    stage1_scan_events_ref s;
  r c%rowtype;
  dups number;
begin
  open c;
  loop
    fetch c into r;
    exit when c%notfound;
  ...
```

# De-duplicating using Row-by-Row Method

...

```
select
  count(*)
into
  dups
from
  stage1_scan_events_ref s
where s.loc_code           = r.loc_code
   and s.rtl_trx_seq_nbr   = r.rtl_trx_seq_nbr
   and s.trx_line_item_seq_nbr = r.trx_line_item_seq_nbr
   and ( s.file_seq_nbr    > r.file_seq_nbr
        or ( s.file_seq_nbr = r.file_seq_nbr
            and s.rowid     < r.rid
            )
        );
```

...

# De-duplicating using Row-by-Row Method

```
...  
    if dups = 0  
    then  
        insert into stage2_scan_events d values (r ...);  
    else  
        r.error_ind := 1;  
        insert into stage1_scan_events_err d values r;  
    end if;  
    commit;  
end loop;  
close c;  
end;
```

# De-duplicating using Array Method

```
loop
  fetch c bulk collect into a limit array_size;
  exit when a.count = 0;
  a_count := a.count;
  for i in 1..a_count
  loop
    select
    ...
    if dups = 0
    then
      null;
    else
      a(i).error_ind := 1;
      insert into stage1_scan_events_err d values a(i);
      a.delete(i);
    end if;
  end loop;
  forall i in indices of a
  insert into stage2_scan_events d values (a(i) ... );
  commit;
end loop;
```

# De-duplicating using Threaded Method

```
cursor c is select
    s.*
    ,cast(null as number(10)) as error_ind
    ,rowid as rid
from
    stage1_scan_events_ref s
where ora_hash(loc_code,threads) = thread;
```

# De-duplicating using Set-based Method

```
alter session enable parallel dml;
```

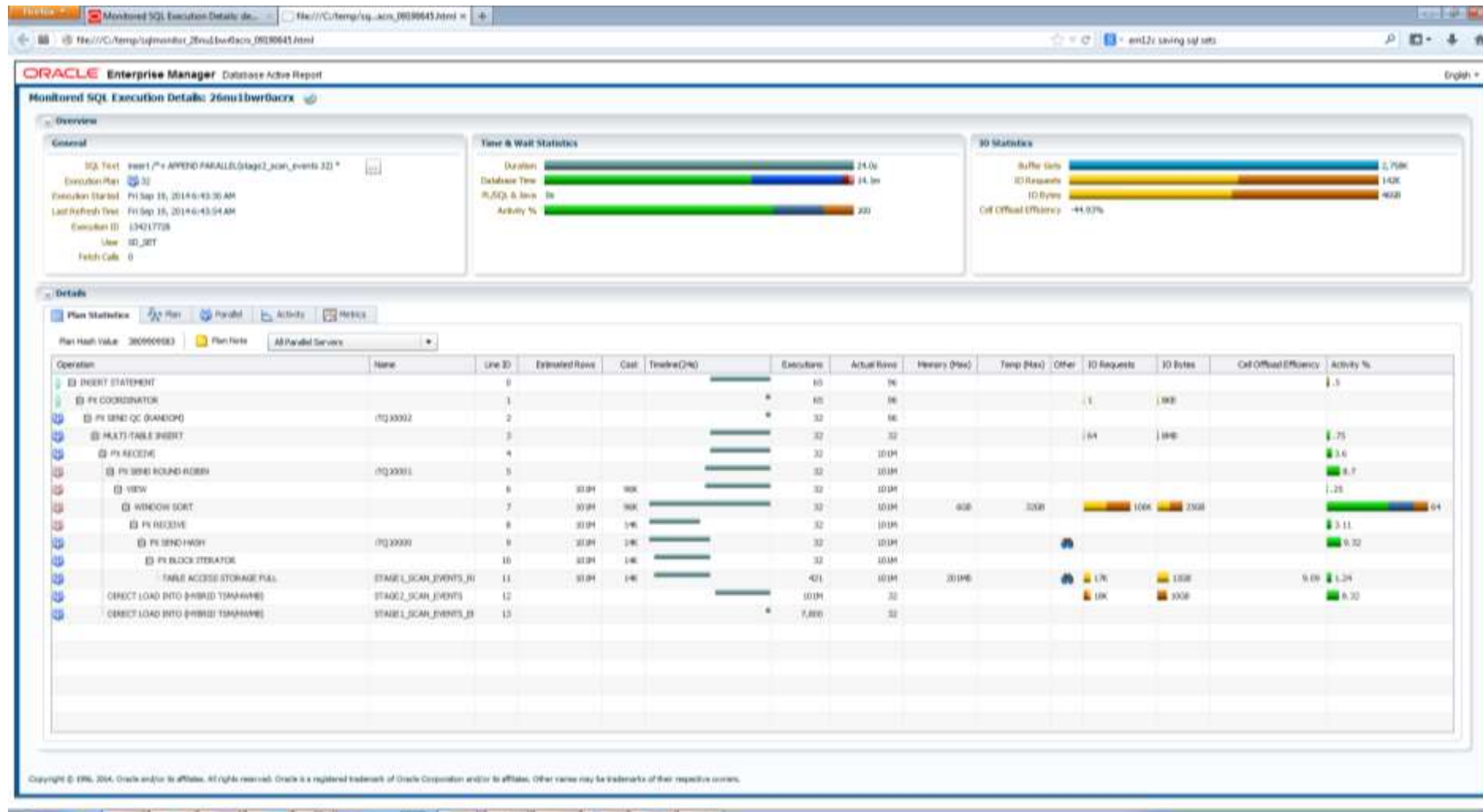
```
insert /*+ APPEND */ first
when error_ind = 1 then into
  stage2_scan_events
values ( ... )
else into
  stagel_scan_events_err
select
  s.*
  ,row_number() over (
partition by
  loc_code,rtl_trx_seq_nbr,trx_line_item_seq_nbr
order by
  file_seq_nbr desc,rowid
) as error_ind
  ,rowid as rid
from
  stagel_scan_events_ref s;
```

```
commit;
```

# De-duplicating



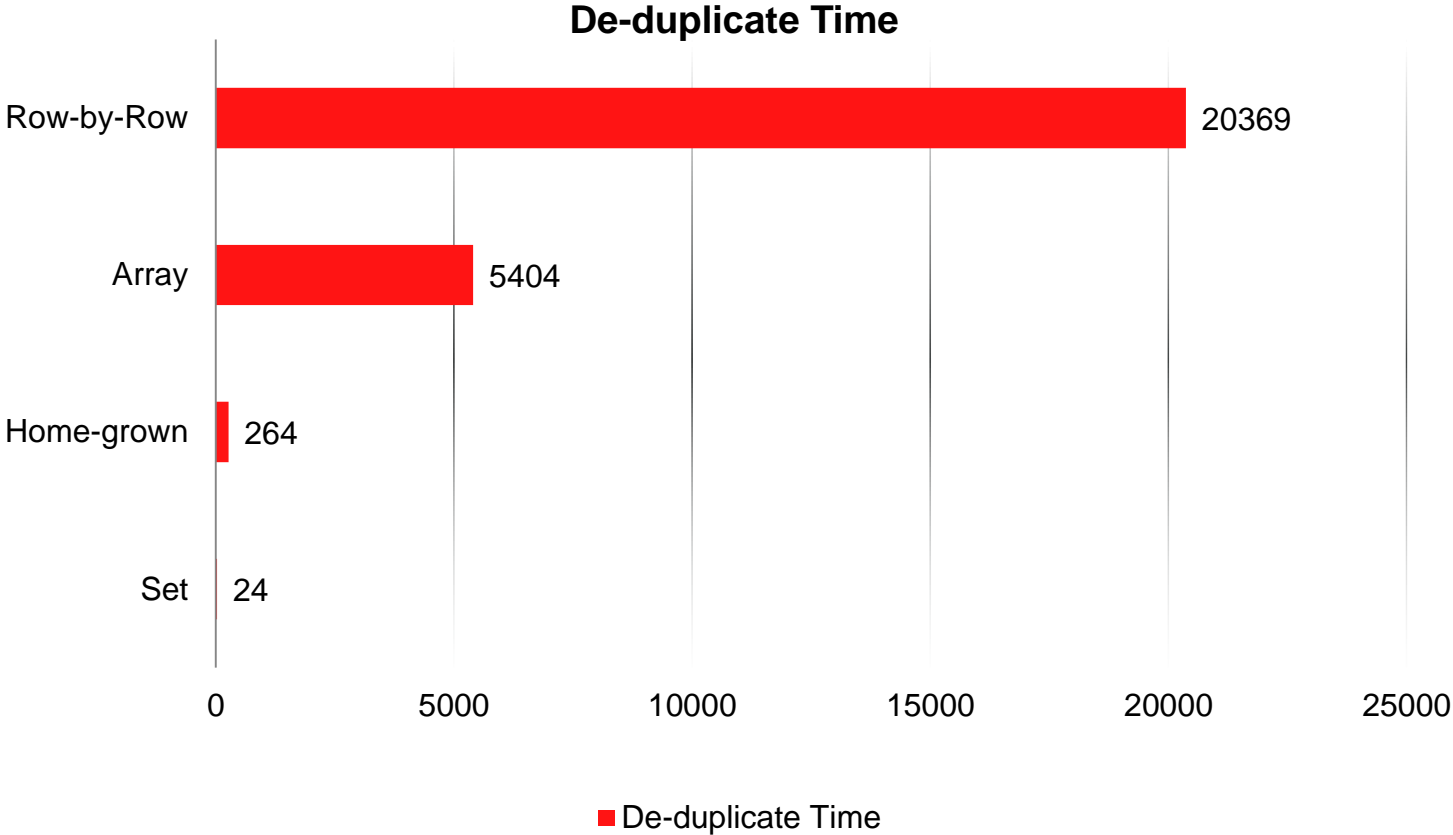
# De-duplicating



# De-duplicating

De-duplicate time in seconds for 100M loaded scan events

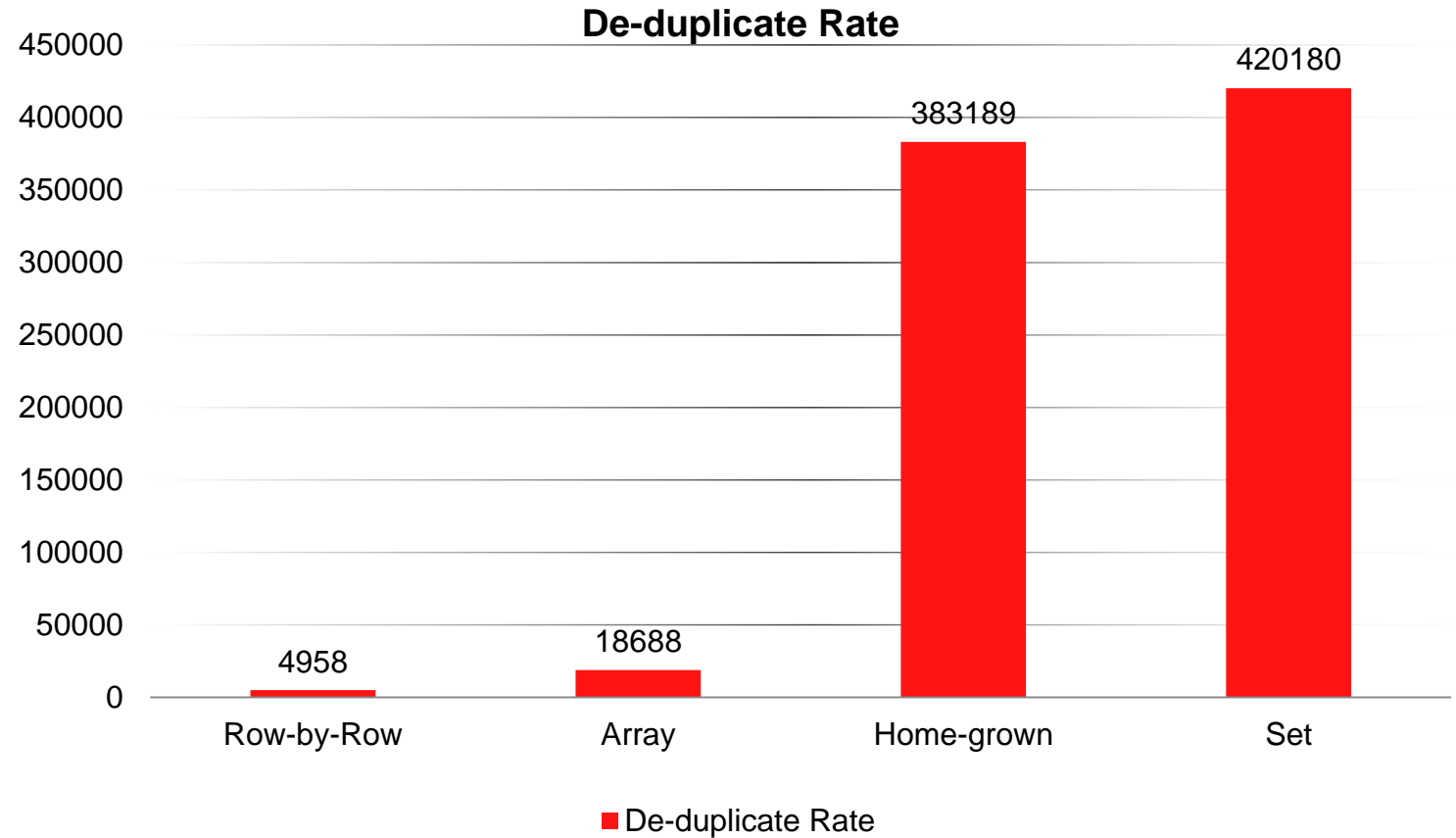
- Array size 100
- Jobs 32
- Degree of Parallelism 32



# De-duplicating

De-duplicate rate in rows per second for 100M loaded scan events

- Array size 100
- Jobs 32
- Degree of Parallelism 32



# Set Programming: Transformation

**ORACLE**  
RWP Video



# Transforming using Row-by-Row Method

```
declare
  cursor c is select
    s.*
    ,cast(null as number(10)) as error_ind
  from
    stage2_scan_events_ref s;

  r c%rowtype;
  dk dim_day.day_key%type;
  lk dim_loc.loc_key%type;
  pk dim_prod.prod_key%type;
  day_code dim_day.day_code%type := '20130922';
  day_key dim_day.day_key%type :=
    sd_convert.day_code_to_key(day_code);
  error_ind number(10);
begin
  open c;
  loop
    fetch c into r;
    exit when c%notfound;
  ...
```

# Transforming using Row-by-Row Method

```
...
    error_ind := 0;
    if r.day_code != day_code
    then
        dk := null;
    else
        dk := day_key;
    end if;
    if dk is null then error_ind := error_ind + 1; end if;
    lk := sd_convert.loc_code_to_key(r.loc_code);
    if lk is null then error_ind := error_ind + 2; end if;
    pk := sd_convert.prod_code_to_key(r.prod_code);
    if pk is null then error_ind := error_ind + 4; end if;
    if error_ind = 0
    then
        insert into stage3_scan_events d values (r ... );
    else
        r.error_ind := error_ind;
        insert into stage2_scan_events_err d values r;
    end if;
    commit;
end loop;
close c;
end;
...
```

# Transforming using Array Method

...

```
if dk is null then error_ind := error_ind + 1; end if;  
lk := sd_convert.loc_code_to_key_rc(a(i).loc_code);  
if lk is null then error_ind := error_ind + 2; end if;  
pk := sd_convert.prod_code_to_key_rc(a(i).prod_code);  
if pk is null then error_ind := error_ind + 4; end if;
```

...

# Transforming using Home-grown Method

...

```
cursor c is select
    s.*
    ,cast(null as number(10)) as error_ind
    ,cast(null as number(10)) as day_key
    ,cast(null as number(10)) as loc_key
    ,cast(null as number(10)) as prod_key
from
    stage2_scan_events_ref s
where ora_hash(loc_code,threads) = thread;
```

...

# Transforming using Set-based Method

```
alter session enable parallel dml;
```

```
insert /*+ APPEND */ first
when error_ind = 0 then into
  stage3_scan_events
values ( ... )
else into
  stage2_scan_events_err
values ( ... )
with
  dim_day_current
as (select * from dim_day where day_code = '20130922')
select
  s.*
,d.day_key
,l.loc_key
,p.prod_key
...
```

# Transforming using Set-based Method

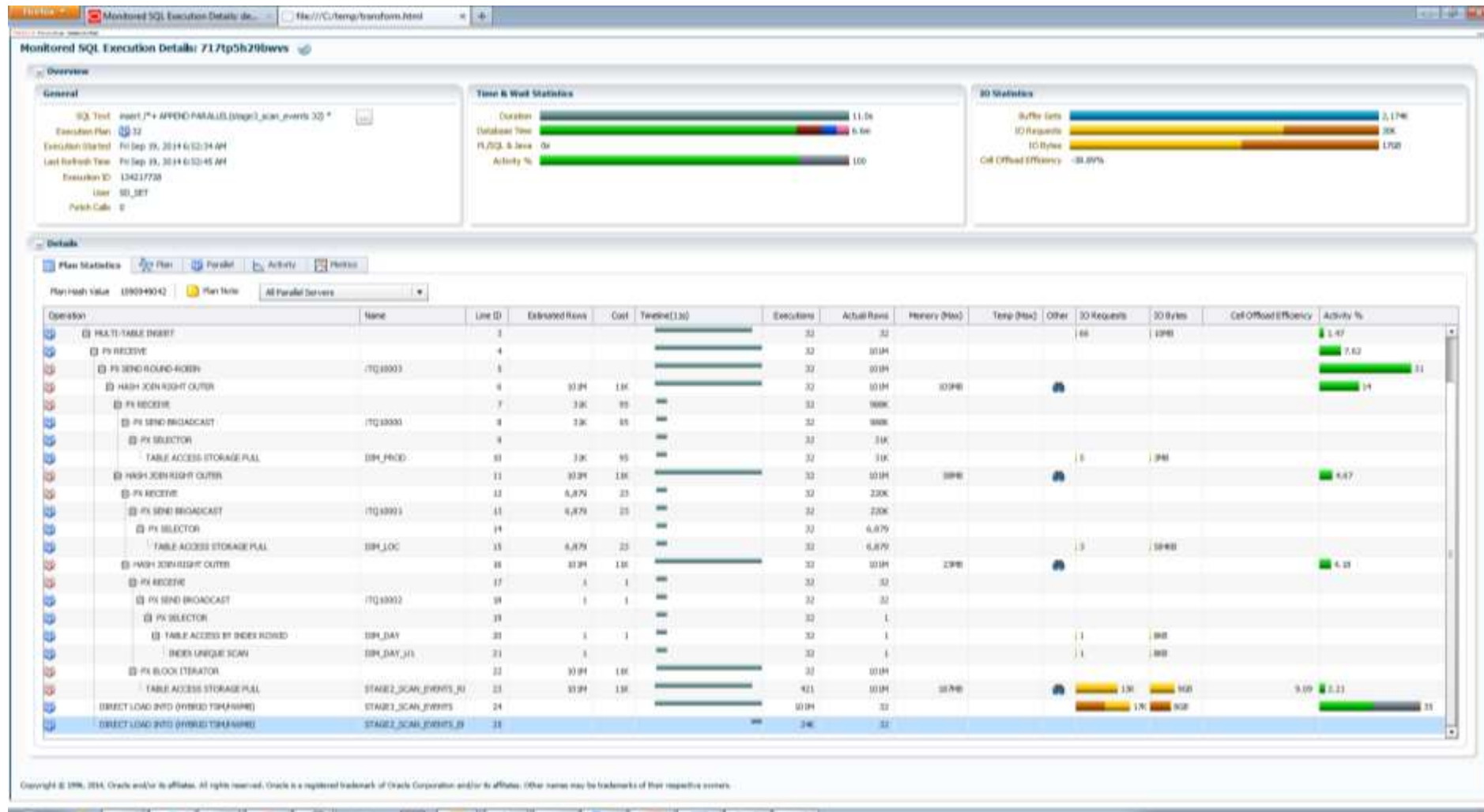
```
...
, (
  case when day_key is not null then 0 else 1 end
+ case when loc_key is not null then 0 else 2 end
+ case when prod_key is not null then 0 else 4 end
) as error_ind
from
  stage2_scan_events_ref s
left outer join
  dim_day_current d
on s.day_code = d.day_code
left outer join
  dim_loc l
on s.loc_code = l.loc_code
left outer join
  dim_prod p
on s.prod_code = p.prod_code;
```

```
commit;
```

# Transforming



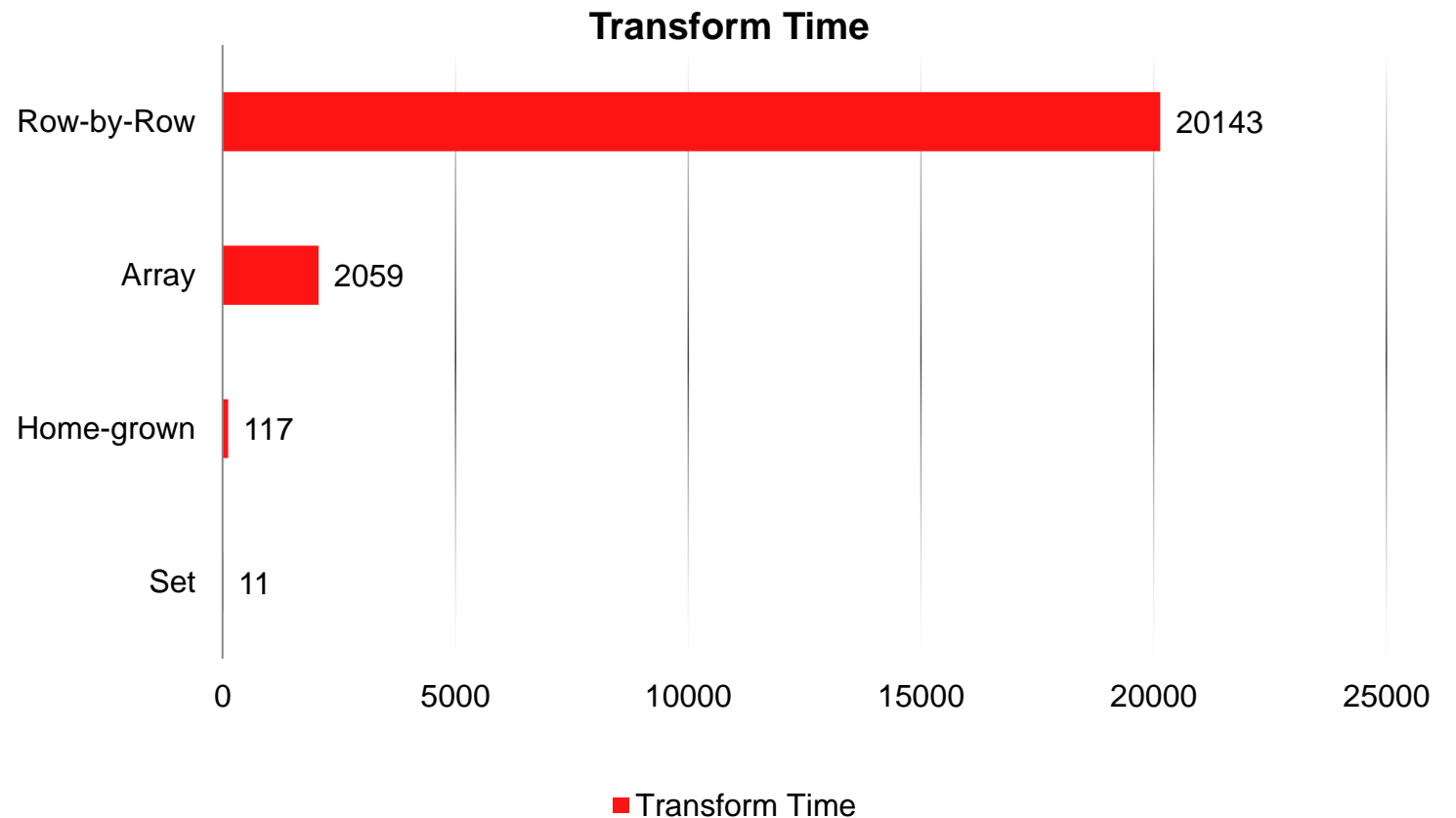
# Transforming



# Transforming

Transform time in seconds for 100M de-duplicated scan events

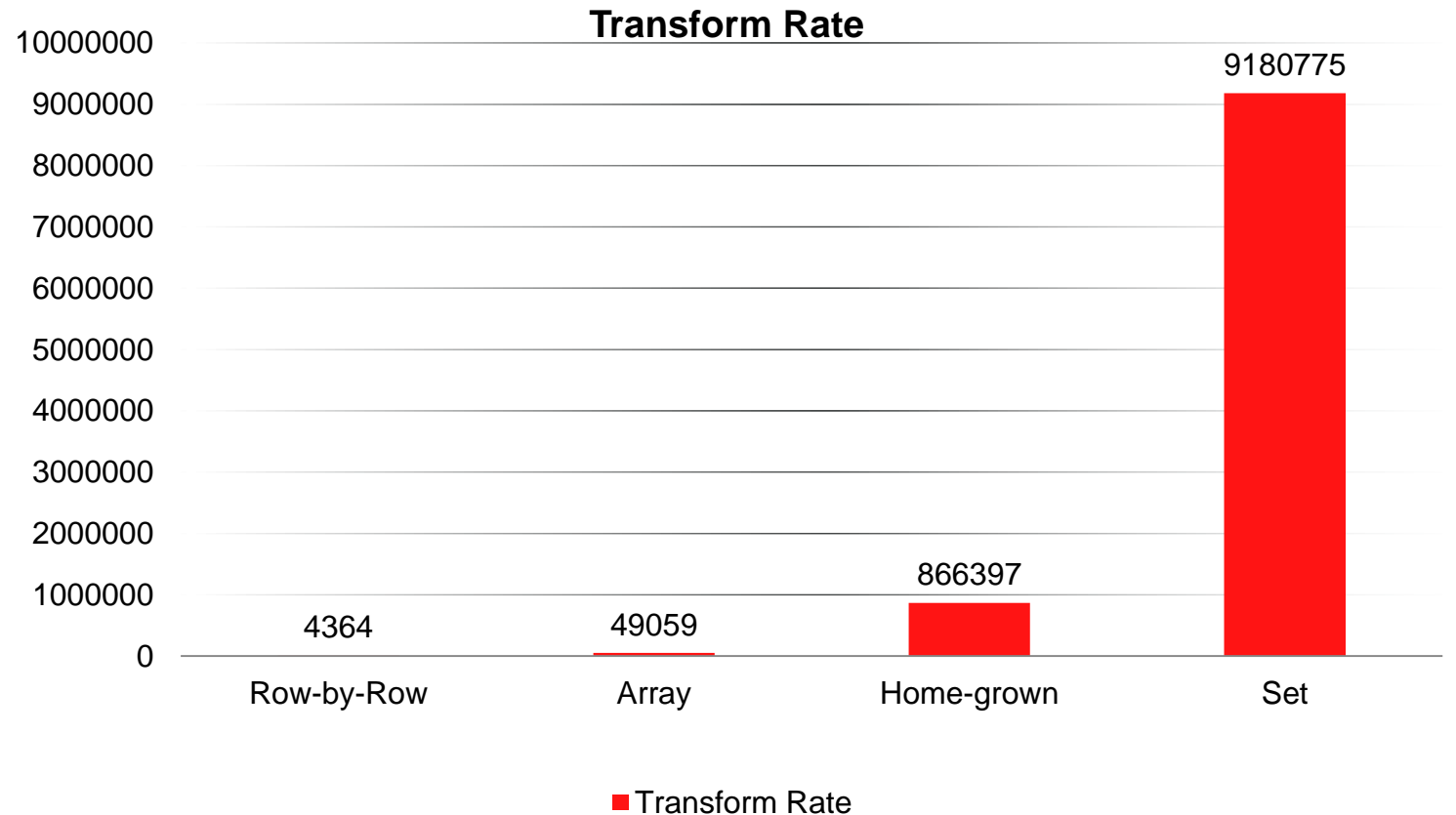
- Array size 100
- Jobs 32
- Degree of Parallelism



# Transforming

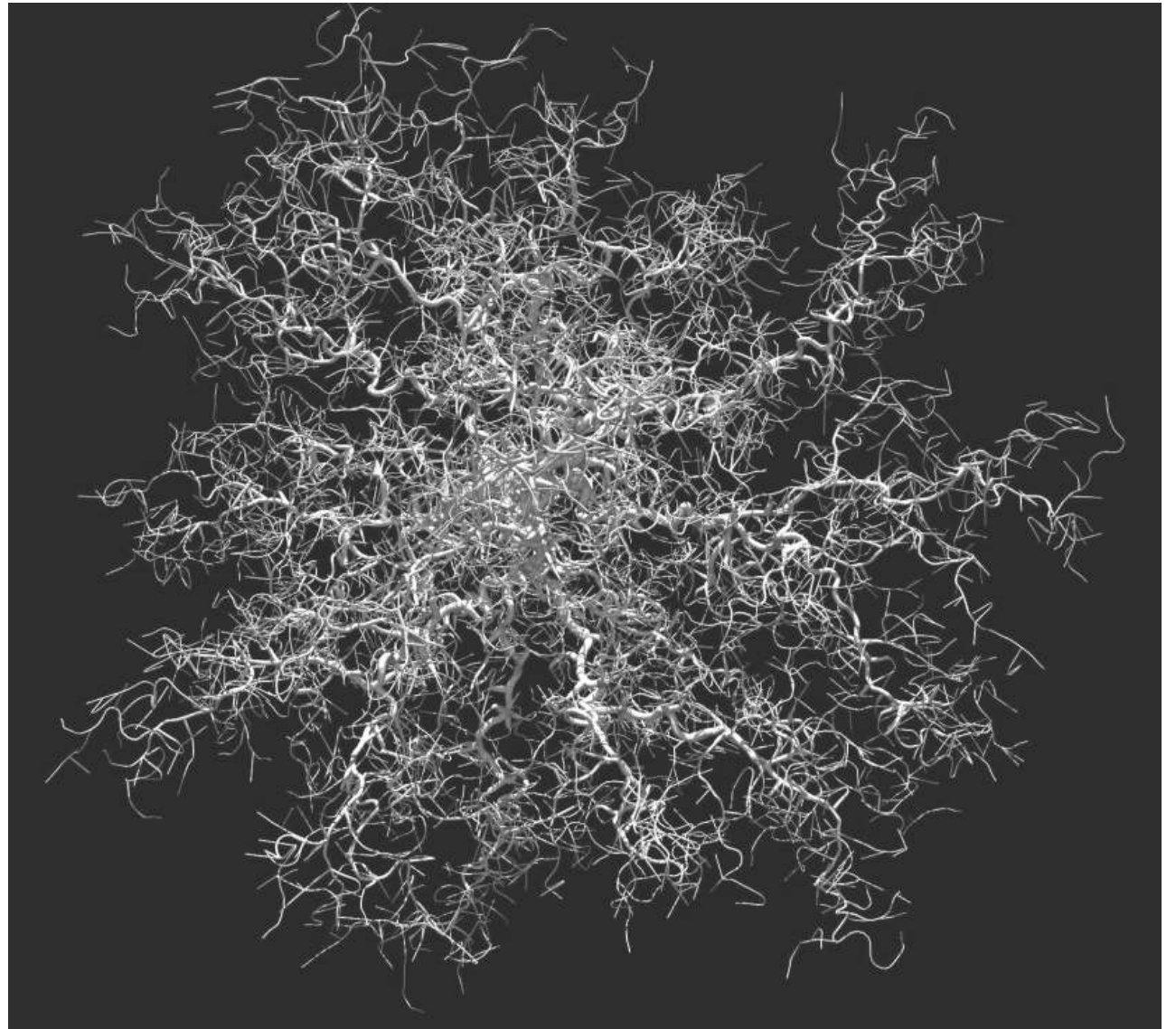
Transform rate in rows per second for 100M de-duplicated scan events

- Array size 100
- Jobs 32
- Degree of Parallelism



# Set Programming: Aggregation

**ORACLE**  
RWP Video



**ORACLE**

**ORACLE**  
REAL-WORLD PERFORMANCE

# Aggregating using Row-by-Row Method

```
declare
```

```
  cursor c is select /*+ INDEX(s) */  
                s.*  
  from  
    stage3_scan_events_ref s  
  order by  
    day_key  
    ,loc_key  
    ,prod_key;
```

```
  r_this c%rowtype;
```

```
  r_last c%rowtype;
```

```
  r_running_total fact_sales%rowtype;
```

```
  ...
```

# Aggregating using Row-by-Row Method

```
...  
procedure update_running_total  
(  
  r c%rowtype  
)  
is  
begin  
  if r.actn_code = 'Sale'  
  then  
    r_running_total.qty :=  
      r_running_total.qty + r.qty;  
    r_running_total.extended_amt :=  
      r_running_total.extended_amt + r.extended_amt;  
    r_running_total.discount_amt :=  
      r_running_total.discount_amt +  
      r.extended_amt * (100 - r.discount_pct) / 100;  
  else  
    r_running_total.qty :=  
      r_running_total.qty - r.qty;  
    r_running_total.extended_amt :=  
      r_running_total.extended_amt - r.extended_amt;  
    r_running_total.discount_amt :=  
      r_running_total.discount_amt -  
      r.extended_amt * (100 - r.discount_pct) / 100;  
  end if;  
end;  
...
```

# Aggregating using Row-by-Row Method

```
...  
procedure start_running_total  
(  
  r c%rowtype  
)  
is  
begin  
  r_running_total.day_key      := r.day_key;  
  r_running_total.loc_key      := r.loc_key;  
  r_running_total.prod_key     := r.prod_key;  
  r_running_total.qty          := 0;  
  r_running_total.extended_amt := 0;  
  r_running_total.discount_amt := 0;  
  update_running_total(r);  
end;  
procedure flush_running_total  
(  
  r c%rowtype  
)  
is  
begin  
  insert into fact_sales values r_running_total;  
  commit;  
  start_running_total(r);  
end;  
...
```

# Aggregating using Row-by-Row Method

```
...
begin
  open c;
  fetch c into r_last;
  if not c%notfound
  then
    start_running_total(r_last);
    loop
      fetch c into r_this;
      exit when c%notfound;
      if      r_this.day_key = r_last.day_key
         and r_this.loc_key = r_last.loc_key
         and r_this.prod_key = r_last.prod_key
      then
        update_running_total(r_this);
      else
        flush_running_total(r_this);
        r_last := r_this;
      end if;
    end loop;
    flush_running_total(r_last);
  end if;
  close c;
end;
```

# Aggregating using Array Method

```
...  
procedure flush_running_total  
is  
begin  
    forall i in 1..a_running_totals.count  
        insert into fact_sales values a_running_totals(i);  
    commit;  
    a_running_totals.delete;  
end;  
procedure buffer_running_total  
(  
    r c%rowtype  
)  
is  
begin  
    a_running_totals(a_running_totals.count + 1) :=  
        r_running_total;  
    if a_running_totals.count = array_size  
    then  
        flush_running_total;  
    end if;  
    start_running_total(r);  
end;  
...  
...
```

# Aggregating using Array Method

...

```
loop
```

```
  fetch c bulk collect into a limit array_size;
```

```
  exit when a.count = 0;
```

```
  for i in 1..a.count
```

```
  loop
```

```
    r_this := a(i);
```

```
    if      r_this.day_key = r_last.day_key
```

```
      and r_this.loc_key  = r_last.loc_key
```

```
      and r_this.prod_key = r_last.prod_key
```

```
    then
```

```
      update_running_total(r_this);
```

```
    else
```

```
      buffer_running_total(r_this);
```

```
      r_last := r_this;
```

```
    end if;
```

```
  end loop;
```

```
end loop;
```

```
buffer_running_total(r_this);
```

```
flush_running_total;
```

...

# Aggregating using Threaded Method

```
alter session enable parallel dml;
```

```
insert /*+ APPEND */ into  
  fact_sales d  
select  
  *  
from  
  table(sd_aggregation.sum_scan_events  
    (  
      cursor(select * from stage3_scan_events_ref s)  
    )  
  );  
  
commit;
```

# Aggregating using Threaded Method

```
create or replace package body
  sd_aggregation
as
function sum_scan_events
(
  c refc
)
return t
pipelined
cluster c by (day_key,loc_key,prod_key)
parallel_enable
(
partition c by hash (day_key,loc_key,prod_key)
)
is
  r fact_sales%rowtype;
begin
  ...
```

# Aggregating using Threaded Method

...

```
loop
```

```
  fetch c bulk collect into a limit array_size;
```

```
  exit when a.count = 0;
```

```
  for i in 1..a.count
```

```
  loop
```

```
    r_this := a(i);
```

```
    if      r_this.day_key = r_last.day_key
```

```
      and r_this.loc_key = r_last.loc_key
```

```
      and r_this.prod_key = r_last.prod_key
```

```
    then
```

```
      update_running_total(r_this);
```

```
    else
```

```
      pipe row (r_running_total);
```

```
      start_running_total(r_this);
```

```
      r_last := r_this;
```

```
    end if;
```

```
  end loop;
```

```
end loop;
```

```
pipe row (r_running_total);
```

...

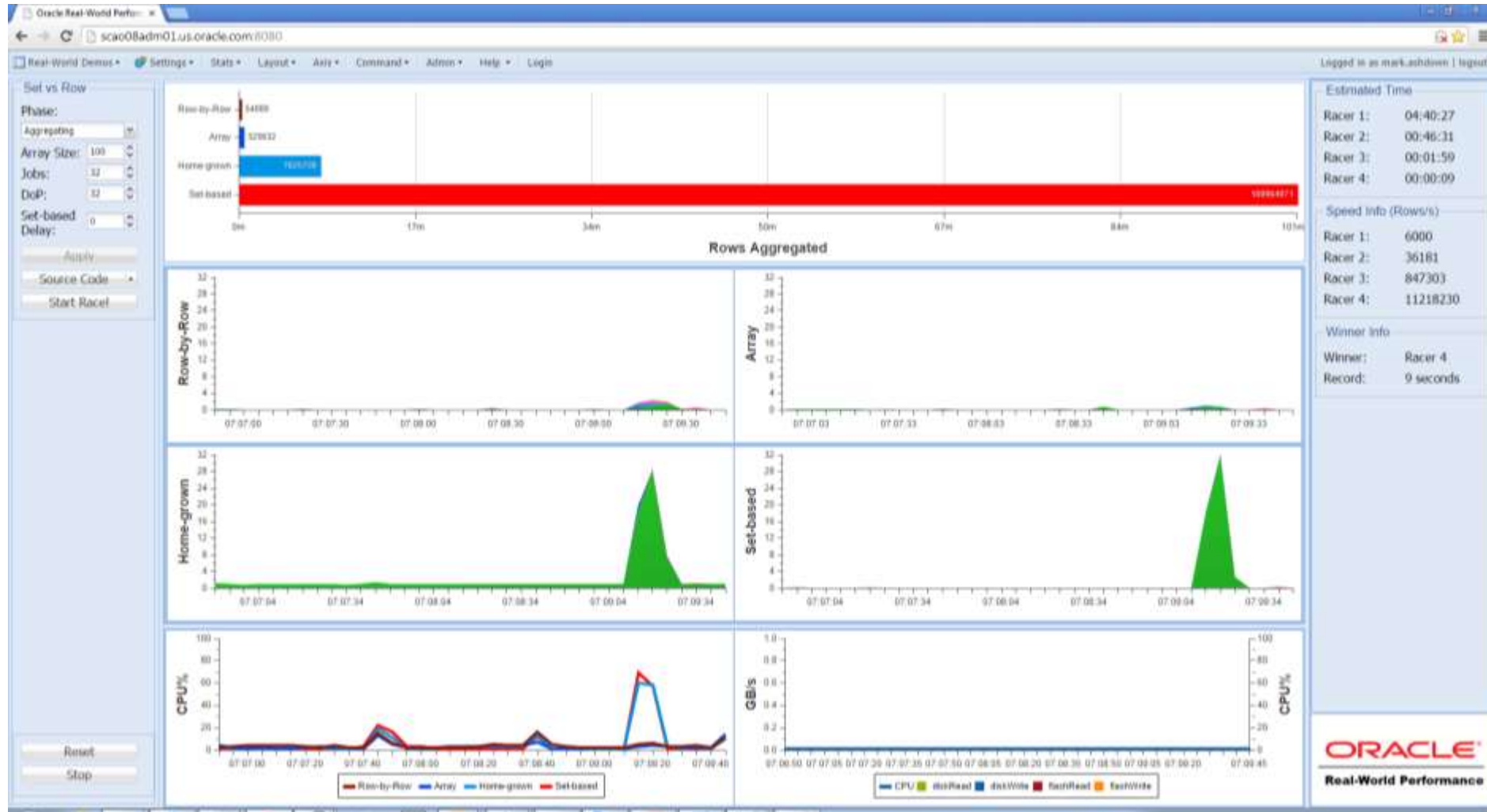
# Aggregating using Set-based Method

```
alter session enable parallel dml;
```

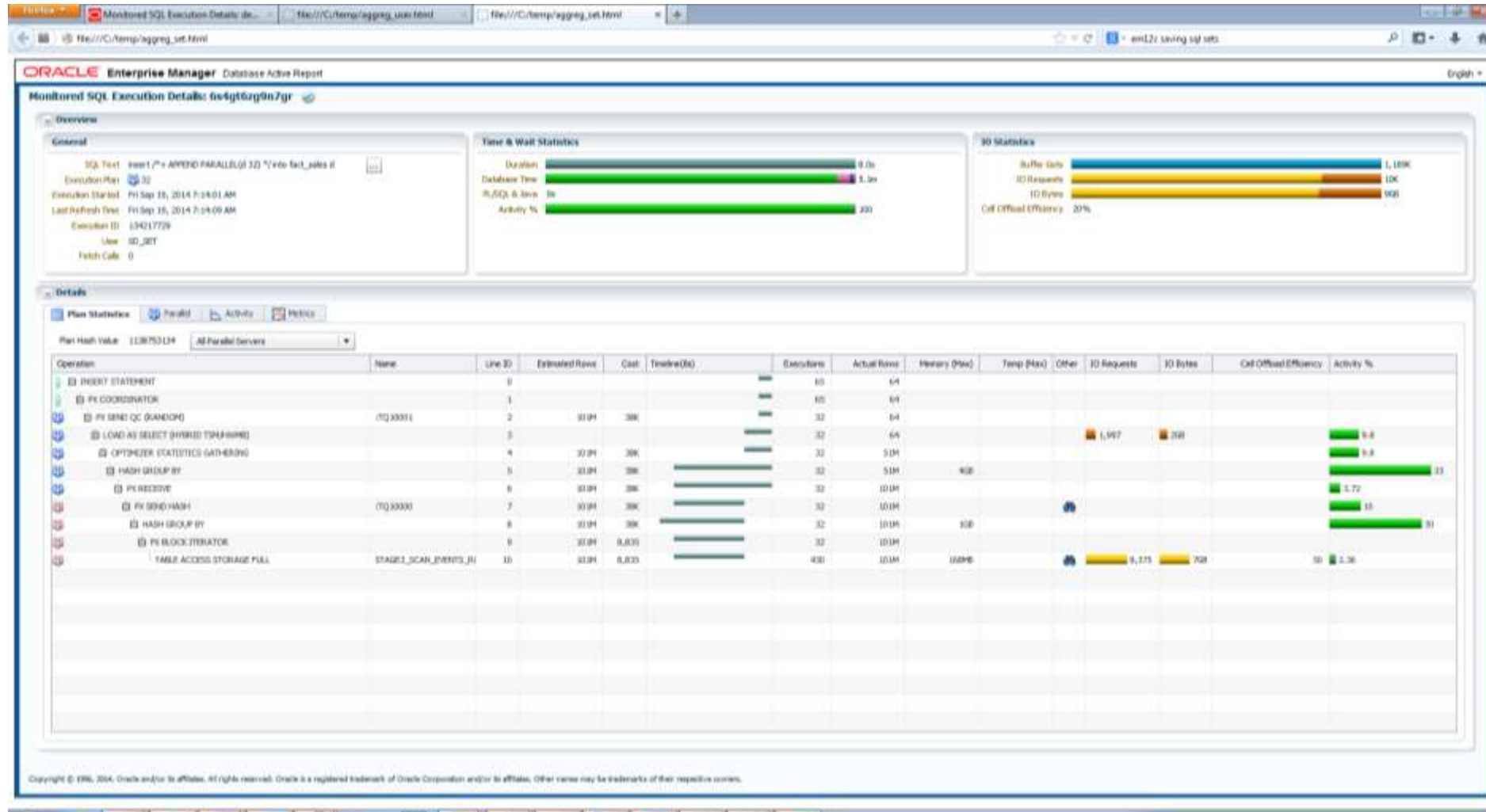
```
insert /*+ APPEND */ into
  fact_sales d
select
  day_key
,loc_key
,prod_key
,sum(case when actn_code = 'Sale' then 1 else -1 end *
      qty) as qty
,sum(case when actn_code = 'Sale' then 1 else -1 end *
      extended_amt) as extended_amt
,sum(case when actn_code = 'Sale' then 1 else -1 end *
      extended_amt * (100 - discount_pct) / 100) as discount_amt
from
  stage3_scan_events_ref s
group by
  day_key
,loc_key
,prod_key;
```

```
commit;
```

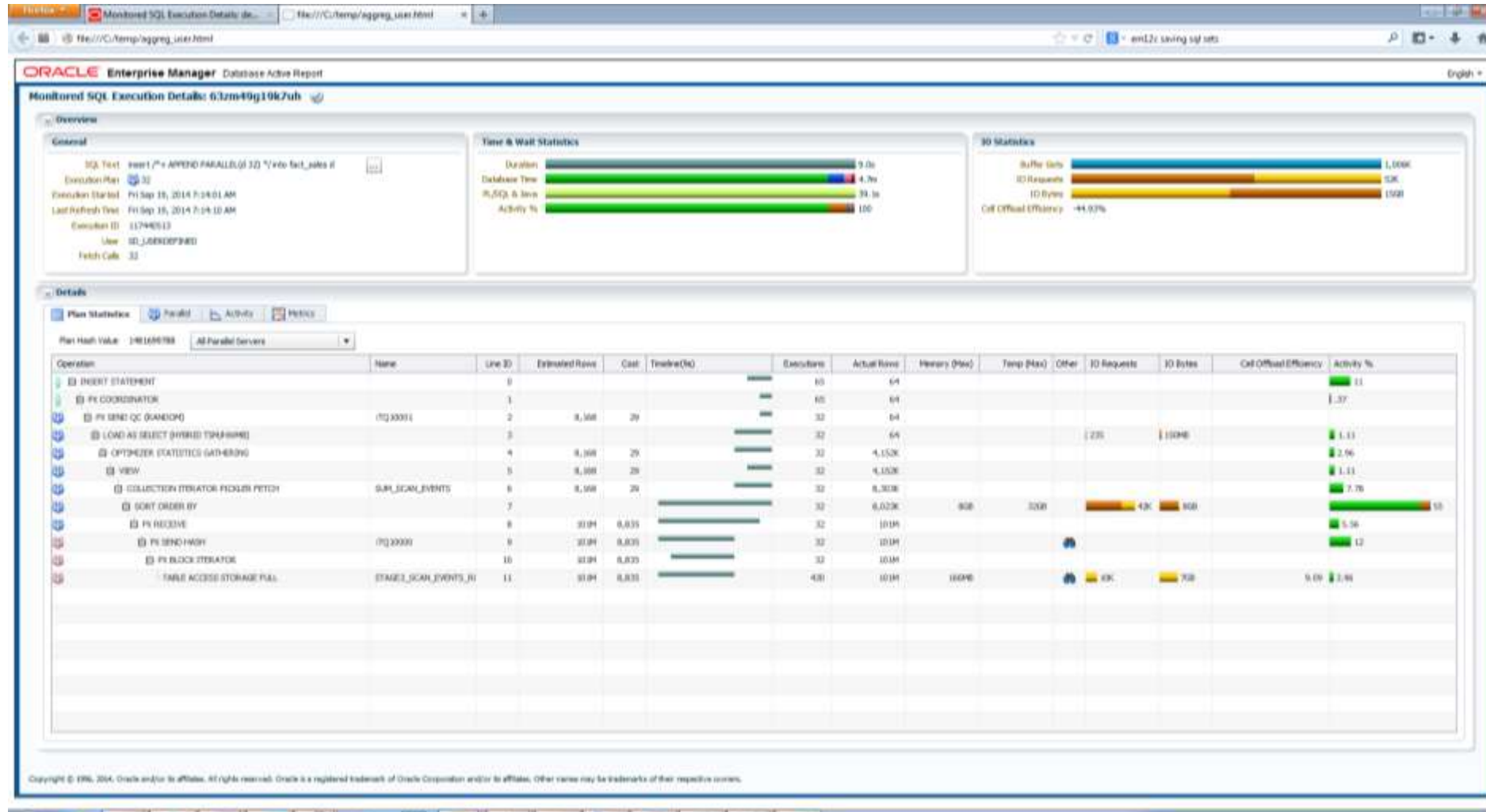
# Aggregating



# Aggregating using Set-based Method



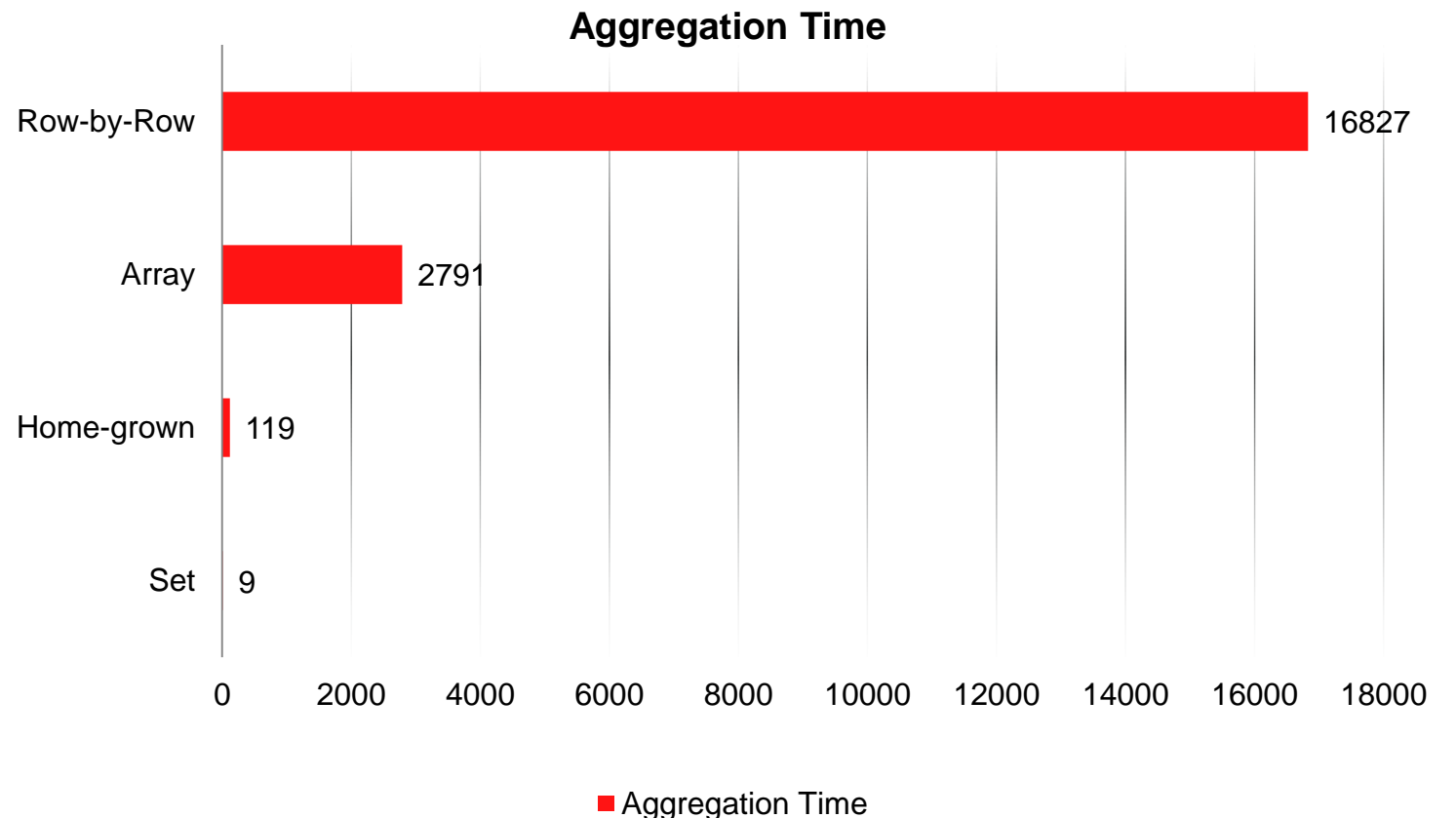
# Aggregating using Pipelined Table Function



# Aggregating

Aggregation time in seconds for 100M transformed scan events

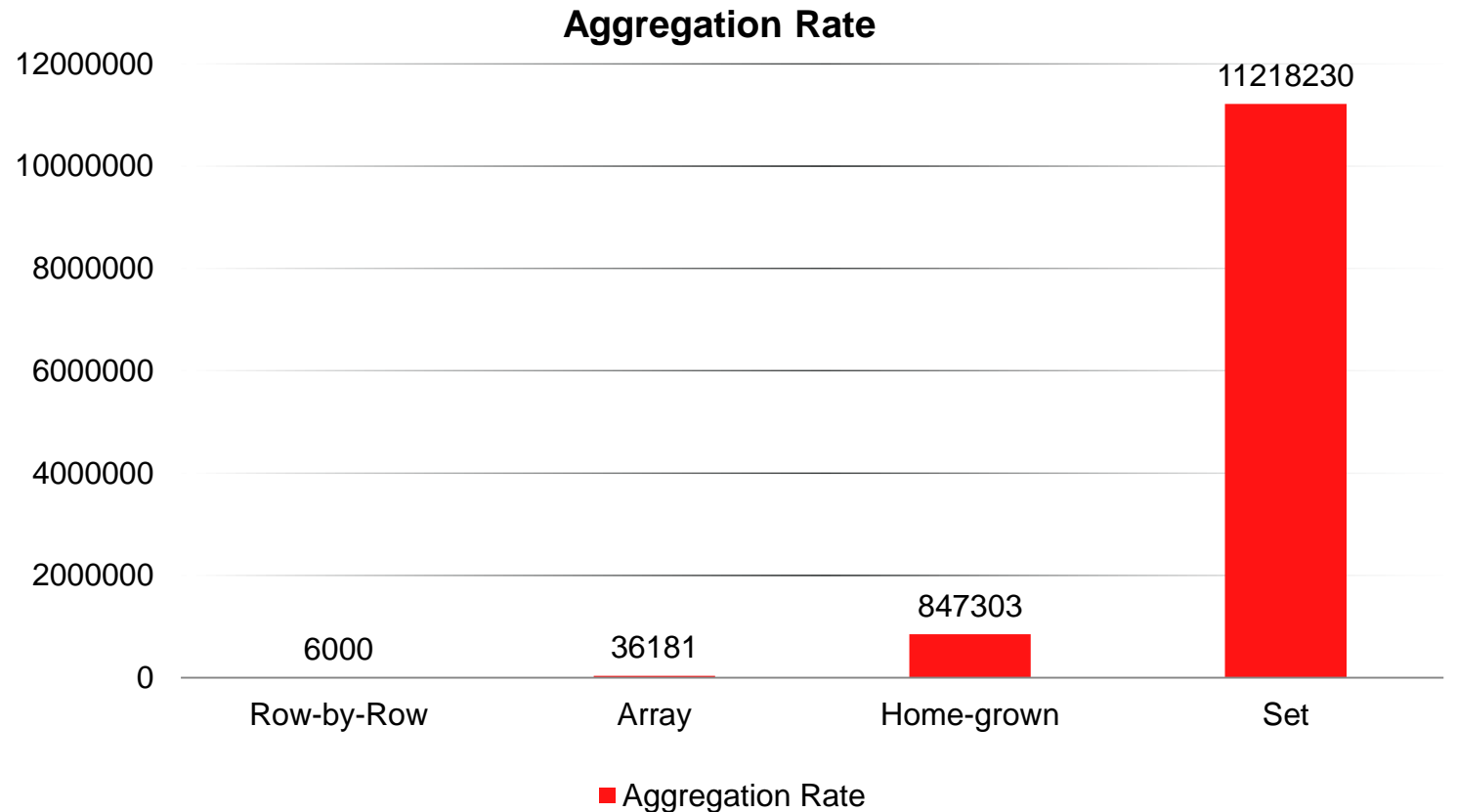
- Array size 1024
- Jobs 32
- Degree of Parallelism 32



# Aggregating

Aggregation rate in rows per second for 100M transformed scan events

- Array size 100
- Jobs 32
- Degree of Parallelism 32

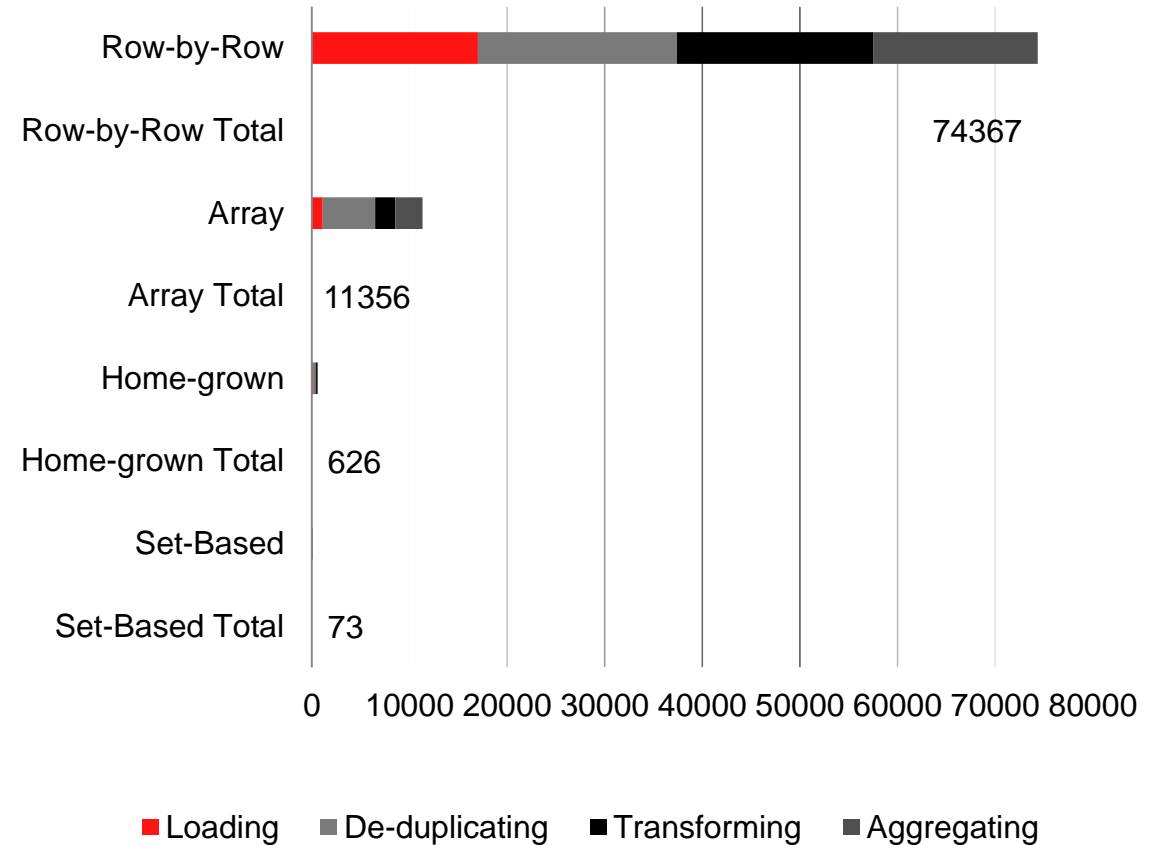


# What Did We Learn?

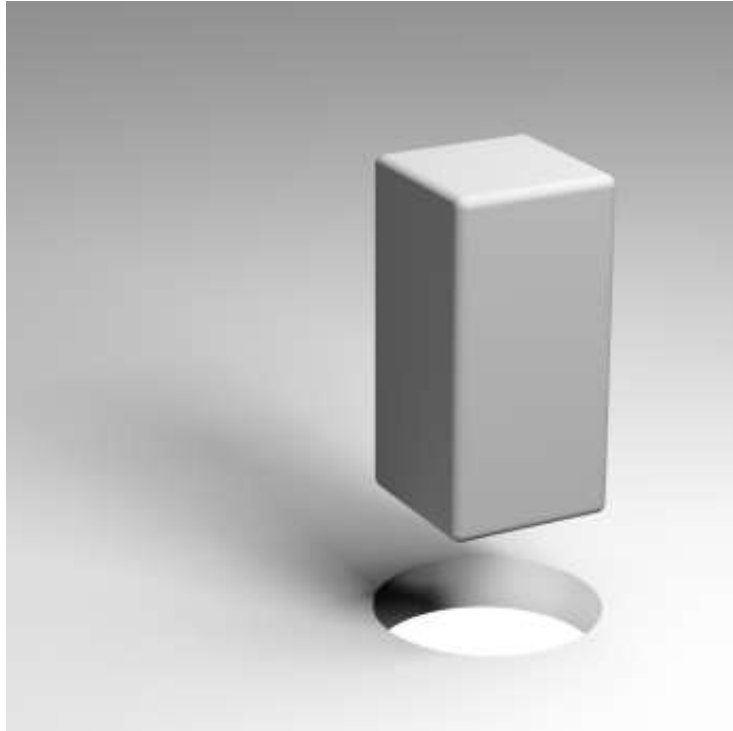


# Summary

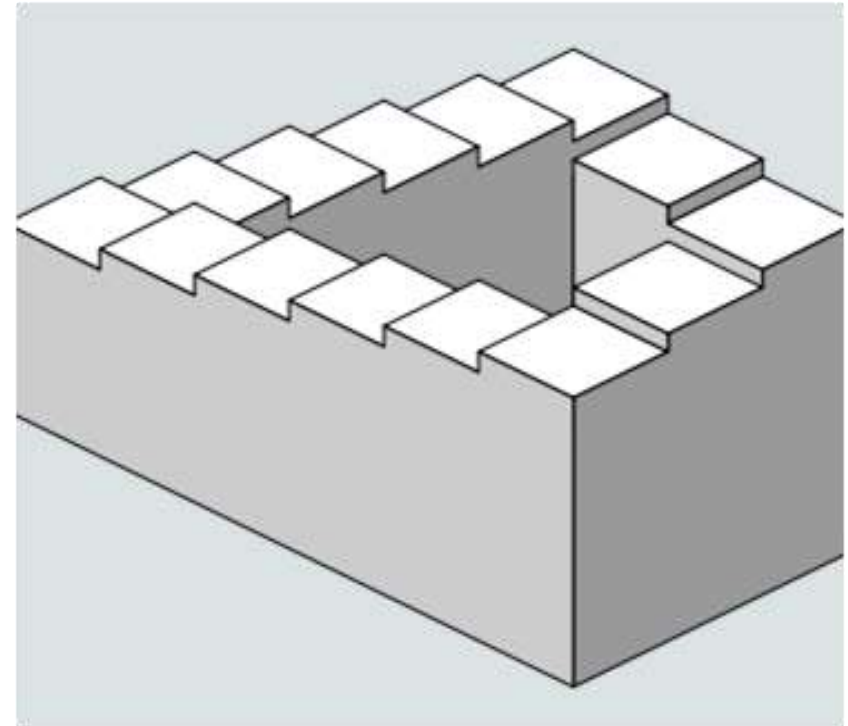
- Summary of processing from raw data to information
  - Array size 100
  - Jobs 32
  - Degree of Parallelism 32



# Root Causes of Suboptimal Database Performance



Row by Row and Array processing for large data sets



Manual Parallelism

# Agenda

- 1 Computer Science Basics
- 2 Schema Types and Database Design
- 3 Database Interface
- 4 DB Deployment and Access Options
- 5 Application Algorithms
- 6 Resource Management

# Resource Management



# Resource Management

## Why bother?

- An unconstrained system can utilize all the resources available in terms of CPU, Memory, Disk or network
- This would have the following consequences:
  - Widely variable response times – not only in the system itself but also other systems sharing the same resources (for example two databases running on the same server)
  - Missing Batch windows
  - Instability of the system (for example too many connections have been seen to cause kernel panic through memory starvation)
- Resource Management Policies and design can mitigate these risks

# Resource Management for Oracle Systems

## Debate

- Why do most Oracle Systems adopt no resource management techniques ?
  - Storage is the ultimate bottleneck, queuing is done in HW
  - Not many users can fully exploit parallel query
- Engineered and modern systems change everything because I/O is not the bottleneck anymore and the challenge moves to CPU and memory allocation
  - It is a new job and needs to be done.
  - There is a requirement to understand the need and implications of what resource management means—system stability and predictability

# Resource Management Principals

- Resource Management
  - Does not make systems run faster
  - Should be used to ensure the system does not run out of control
  - Makes some jobs run slow so that other jobs can run fast
- In Summary - Resource Management
  - Is NOT a turbo
  - Is a gatekeeper

# Resource Management

## Tools and Options

- Non-Oracle techniques
  - Middleware or tool queuing
  - Performance implications when using MW to parallelize DW operations
  - Virtualisation
- Database Services
  - Course grain resource management
  - Assign workloads to specific RAC nodes
    - Load separate partitions on each node
    - Separate query workloads

# Resource Management

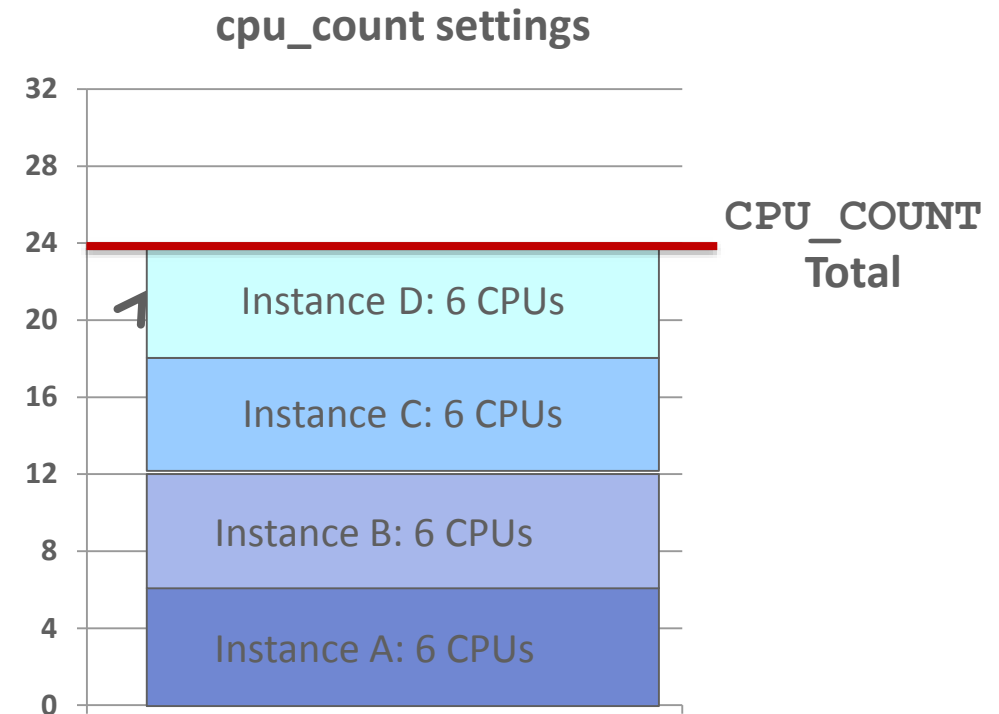
## Tools and Options

- Database Resource Manager
  - Instance caging
    - Limit Oracle instance to number of CPU cores
    - CPU\_COUNT parameter
  - Resource Consumer Groups, which allow you to control resources such as
    - Max DoP
    - Percentage of Parallel Servers
    - CPU
  - IO resource manager (Exadata only)
    - Use other RM tools first
    - Can be used for systems with many databases
    - Only relates to disk IO not Flash Activity

# Resource Management Instance Caging

## Setting `cpu_count`: Partition Approach

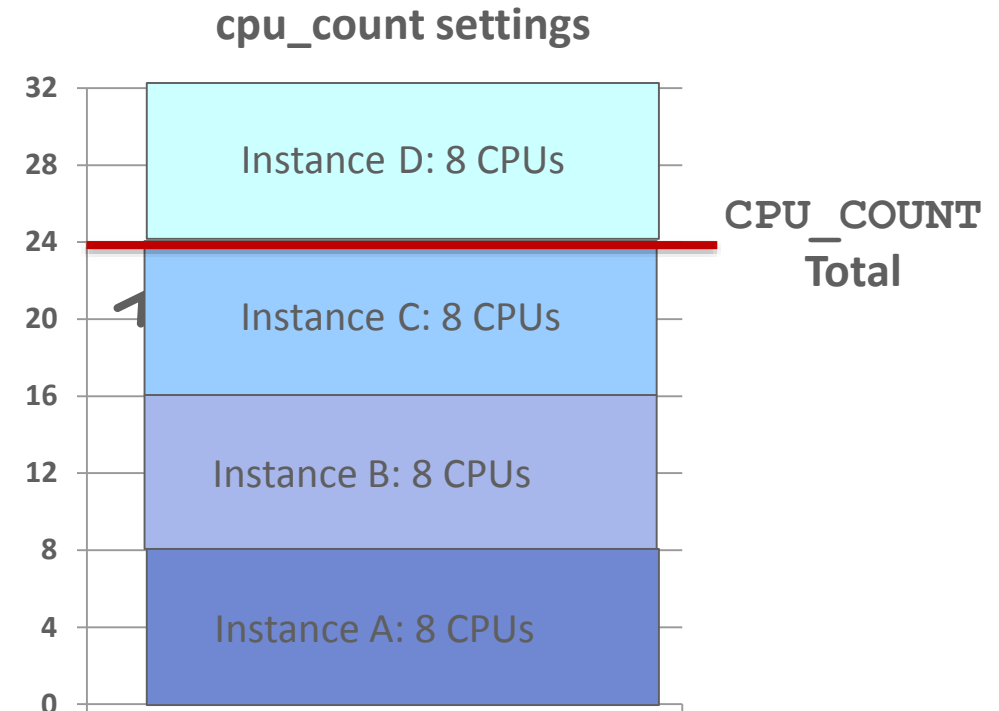
- Partition CPUs among the database instances
  - $\text{sum}(\text{cpu\_counts}) \leq \# \text{cpu threads}$
- Partitioning provides maximum isolation
  - No CPU contention between instances
  - But if one instance is idle, its CPU allocation is unused
- Best for performance-critical databases



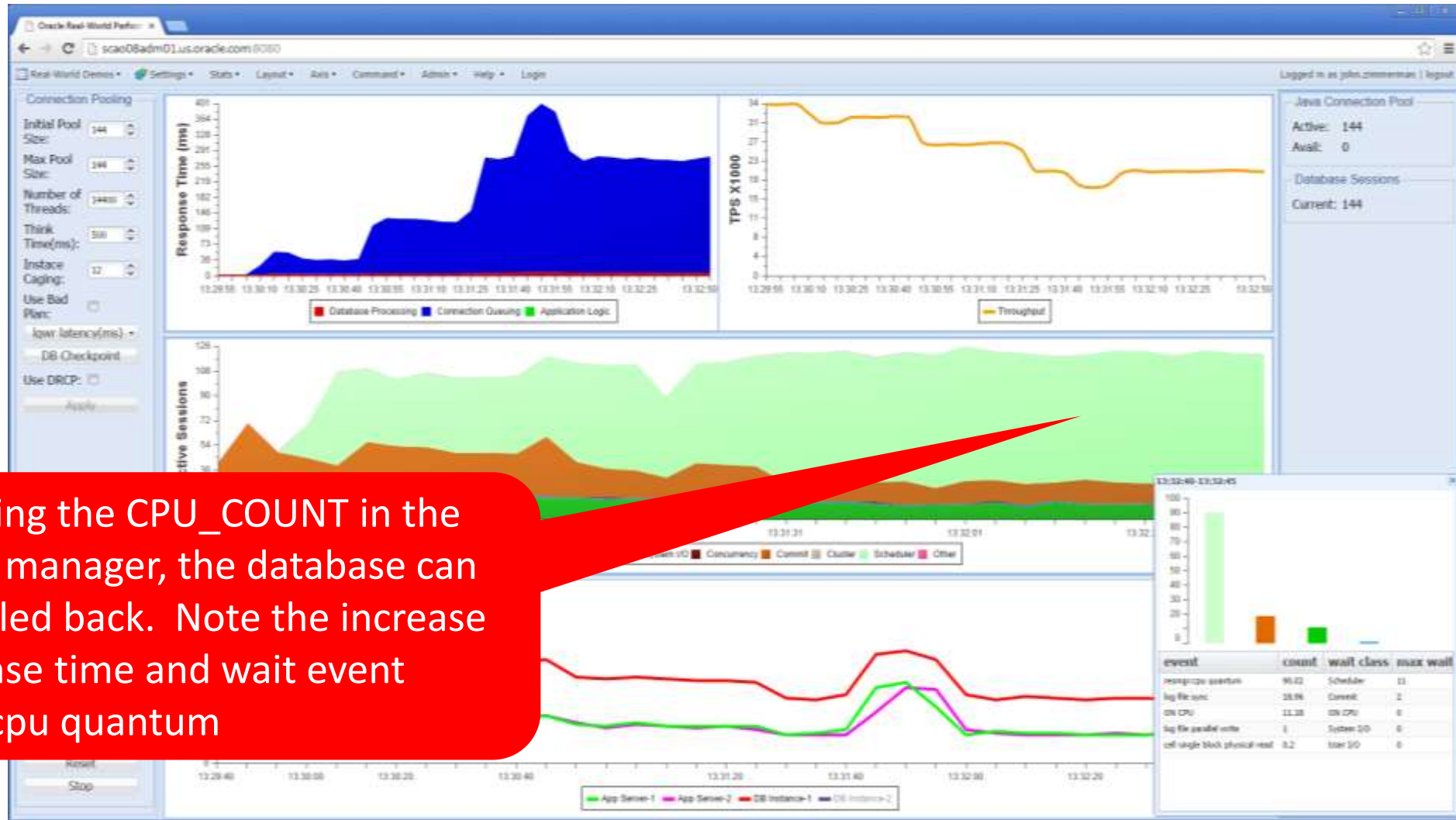
# Resource Management Instance Caging

## Setting `cpu_count`: Over-Subscribe Approach

- Over-subscribe the CPUs among the database instances
  - $\text{sum}(\text{cpu\_counts}) \leq 3 \times \# \text{cpu threads}$
  - Monitor CPU utilization to see if there's room!
- Over-subscribing provides efficient CPU utilization
  - Some contention for CPU if databases are sufficiently loaded
  - Contention is controlled, so system is still stable
- Best for non-critical databases



# Resource Management - Instance Caging



# Resource Management Policies

- The higher up the Software/Hardware stack resource management is performed the more effective it will be at preventing overload and allow effective prioritization.
- For Database Resource management a hierarchy of resource management activities is appropriate:
- High Level
  - Middle Tier queuing and prioritization
- Mid Level
  - Database Services
- Low Level
  - Database Resource Manager
- Very Low Level
  - I/O resource Management

# Resource Management Policies

- Having designed a resource management policy it is almost impossible to prove it works
  - Survival of a system is not good evidence
  - Response times completing within SLA is not good evidence
  - Have you ran a controlled experiment with and without resource management
- Remember that for resource management to be effective under load something has to slow down for other workloads to succeed.

# Main Tools for System Wide Database Diagnostics

- AWR Report
  - Diagnostic information on the instance or database level
  - Information for the system as a whole
- ADDM Report
  - Recommendations based on AWR information
- ASH Report
  - System wide or granular information such as the session level

# Main Tools for Single User or SQL Statement Diagnostics

- Explain Plan
  - Execution plan for a single SQL statement
- SQL Monitor Report
  - Execution plan and diagnostic information for a single SQL statement
- 10046/SQL Trace
  - Diagnostics information for a single session
- ASH Report
  - System wide or granular information such as the session level

# Hardware and Software Engineered to Work Together

ORACLE®